The Pennsylvania State University

The Graduate School

College of Engineering

STATE SPACE MODELING, ANALYSIS, AND SIMULATION OF IDEAL SWITCHED RLCM NETWORKS

A Thesis in

Electrical Engineering

by

Saleh Mahdi Albeaik

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2014

The thesis of Saleh Mahdi Albeaik was reviewed and approved* by the following:

Jeffery S. Mayer
Associate Professor of Electrical Engineering
Thesis Advisor

John D. Mitchell
Professor of Electrical Engineering

Kultegin Aydin
Professor of Electrical Engineering
Head of the Department of Electrical Engineering

*Signatures are on file in the Graduate School

ABSTRACT

The design process for power converters and electromechanical drives would be enhanced by the availability of modeling software that represents the fundamental nature of these systems, namely, switch-mode operation of the semiconductor devices. Established network simulation programs, such as SPICE and its many descendants, utilize a combination of companion models and modified nodal analysis (CM + MNA) that permits very simple implementations of algorithms for simulating the dc, ac, and transient response of most electrical networks. In the early days of computer modeling, the simplicity of these algorithms was critically important while the attention being given to networks with switch-mode operation was modest and focused on digital logic. Thus, SPICE (CM + MNA) became the de facto standard for network simulation. It can be used for transient simulation of power converters, but it is inherently slow and can provide little insight beyond the raw time-domain response of the converter. These limitations can be addressed by replacing CM + MNA with state space modeling (SSM).

Ideal switches, the main subject of this thesis, impose computational difficulty to simulation software. The ideal behavior is extremely non-linear, potentially causing impulses to exist in the network. Moreover, to employ piecewise linear analysis techniques, switch conduction state dependencies must be resolved. The thesis offers a computationally efficient method to handle the presence of impulses as an alternative to more complex approaches presented in literature. Analyzing the presence and the effect of impulses efficiently makes resolving switch state dependencies using the common iterative search algorithm feasible.

Simulation software based on this thesis was designed and implemented. An overview of the software design is presented towards the end.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Introduction

1.1 Motivation

We are living in an era of ever faster and more sophisticated products and services. To design and build systems that meet the standards of this era, engineers usually must conduct complex analyses of system performance using mathematical models. Computer systems, both hardware and software, are among the technologies with the fastest development process and provide a basis for conducting analyses of many other systems. More pointedly, object-oriented programming languages and their associated design processes have emerged as key elements for creating engineering analysis and design tools that are critical to increasing engineering productivity.

Programs for simulating electronic circuits were among the limited set of applications developed during the first few generations of digital computer technology. The most successful of those programs was SPICE – Simulation Program with Integrated Circuit Emphasis – which went on to become a de facto standard in the integrated circuit industry and in electrical engineering more generally. The success of SPICE (including its descendants) was due in large part to the fact that the combination of so-called companion models and modified nodal analysis (CM+MNA) that it uses is highly adapted to simulating electrical networks. The simplicity of the CM+MNA was particularly valuable with early computers, and simulation, as opposed to other forms of analysis, was of paramount importance.

Having powerful programming languages and fast hardware readily available, it is useful to reconsider the modeling and simulation techniques that did not gain popularity in the earlier days. The objective of this research is to design and implement software to model and simulate switched RLCM networks using state space methods and ideal switch models, as these methods and models are more appropriate for the analysis of power converters.

1.2 Problem Description

There are several practical methods for modeling and simulating so-called RLCM networks that are comprised of resistors, inductors, capacitors, and mutual inductance as well as independent sources. The addition of switches in an RLCM network, however, poses a significant challenge. In particular, SPICE and its many descendants utilize a combination of companion models and modified nodal analysis (CM+MNA) that can accommodate highly accurate behavioral modeling of diodes and transistors but does not recognize changes in network topology arising from the use of ideal switch models. When analyzing power converters at a system level; however, it is often desirable to model the switching devices as being ideal and to consider the sequence of topologies that the converter goes through. Thus, state space modeling (SSM) is a preferred alternative to CM + MNA.

In recent years, several SSM-based simulators have been proposed and implemented. However, many of these are limited to particular types of networks as the result of simplifying assumptions. These assumptions commonly fall under two main categories: (1) suitability of state space formulation to computer environments and (2) efficiency in handling the transitions between two consecutive topologies [1], [2], [3]. The first category of assumptions has led to simulators that are reliant on pre-generated state transition matrices to model the network. The second category of assumptions includes limiting the mutual behavior of switches to reduce the complexity of algorithms for determining transitions between topologies. Examples include relying on equivalent multi-terminal switches to hide dependencies, limiting the number of switches that may change state at any particular time instant, and assuming continuity of capacitor voltages and inductor currents.

This thesis project utilizes modern programming approaches and modern analysis methods to obviate the need for restrictive assumptions in state space modeling and simulation of switched RLCM networks. The use of object oriented programming languages facilitates formulating state space models and handling changes in network topology. Each topology corresponds to a linear electrical network, which can be solved using numerical integration methods. The generated solutions can be monitored automatically to detect conditions referring to change in the state of any of the switches. At this instant, the simulator evaluates the state of each switch in order to generate the new topology and the corresponding state space model. This approach eliminates the need for a prior knowledge of all possible state space models or state transition matrices. The automation of the process also reduces the complexity and the time associated with finding the proper next topology.

A software application for modeling and simulating switched networks has been designed and implemented. The use of ideal switch models in this application is of particular importance for simulating system level behavior of power converters. Moreover, the use of state space modeling methods allows the use of the application for purposes other than simulation. For example, studying stability of systems without the need to simulate the response of the network.

1.3 Technical Challenges

To simulate switched RLCM networks using state space modeling techniques, we must first address a number of challenges. Ideal switches have nonlinear i-v characteristics, which can disturb the network. The nonlinearity of ideal switches takes effect only at transition instants. This makes it possible to reduce the simulation problem to analyzing a sequence of linear RLCM networks. However, analyzing switch transition instants is challenging.

The first challenge to analyzing switch transition instants is identifying these instants. This challenge is specific for networks that have internally controlled switches. For numerical simulation applications, this challenge is commonly addressed using numerical binary search algorithms. The instants of switch transitions are identified in an iterative process attempting to specify the time instant at which a switch violates its conditions. The accuracy of identifying the time instant can be controlled by the number of iterations executed.

The instantaneous response of ideal switches can cause discontinuity in capacitor voltages or inductor currents. The new capacitor voltages and inductor currents must be calculated, which is the second challenge. This challenge is commonly called the re-initialization problem, or the problem of finding consistent initial conditions. In the literature, re-initialization has been addressed in multiple ways. A popular approach was proposed by [4] based on two-step integration algorithm. However, their approach requires special integration algorithms, which adds to the complexity of the solution. Another approach was proposed recently by [5] based on an energy jump rule. While that method is relatively easy to implement, finding the consistent initial conditions requires computing a Moore-Penrose pseudo inverse and finding the null space of a matrix. These two linear algebra operations are not the simplest to calculate, and are supported natively by only a limited number of programming languages. In particular, C++, which is the language used for this project, does not support it natively.

When a switch transitions, it is possible that it will trigger a sequence of other switch transitions throughout the network. Therefore, whenever the network has experienced a switch transition, a thorough analysis must be performed to identify the correct topology the network will enter after the initial event. A recent work by [6] addressed this challenge by constructing closed form solution. However, their approach assumes that capacitor voltages and inductor currents are continuous at the transition instant. An earlier work [4] used an iterative approach. The iterative approach traces impulses, while freezing time, to identify the sequence of switch transitions. However, that approach also requires performing some analysis to detect the presence of analytical impulses and to calculate their areas.

In this project we were interested in a general and automatic simulation process that does not require prior knowledge about the network dependencies nor user intervention. Some of the issues with the approaches in literature will be addressed in this thesis before delving into the software design of the simulator.

1.3.1 Transitions in Switch Conduction States

Transitions in the conduction state of an ideal switch (may) force instantaneous changes in the values of network variables. Network variables associated with storage elements are particularly sensitive to such changes due to the time derivatives in the i-v characteristics of the storage elements. This means that an abrupt change in the network initiated by a switch can introduce impulses in the network. To illustrate this process, consider the circuit shown in Figure 1.1. The switch S1 changes state from OFF to ON at t_o.

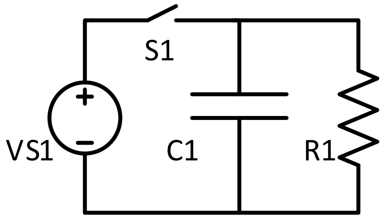


Figure 1.1 Switched active RC network

This forces the voltage between the switch terminals to zero instantaneously. The network voltages and currents must continue to satisfy KVL and KCL, but the capacitor is the only element in the loop (VS1, S1(ON), C1) whose voltage may change. Consequently, the capacitor voltage undergoes a step change with an associated current impulse ($i_C = C \frac{d}{dt} v_c$).

The changes in network at a switch transition can be formalized as follows:

- A voltage or current changes instantaneously due to a trigger event. Trigger events considered in this work include switch transitions and discontinuities in independent sources.
- A switch transitions in one of two ways:
    - Switch turning ON
        - The voltage change across the terminals of the switch steps to zero forcing an instantaneous change in voltage around the remainder of any loops that include the switch as in Figure 1.2.
    - Switch turning OFF
        - The current change through the switch steps to zero forcing an instantaneous change in the current through the remainder of any cut-sets that include the switch as in Figure 1.3.
- To maintain KVL and KCL, one of the following three conditions must occur in the affected loops or cut-sets:
    - An element (or elements) simply absorbs the change without instigating changes in variables for any other element as in Figure 1.4.
    - A switch transitions between ON/OFF states, possibly instigating even more switch transition events. An example of this scenario is the switching events in a buck converter, shown in Figure 1.5.
    - A capacitor or inductor absorbs the change thereby creating a current or voltage impulse as in Figure 1.1.

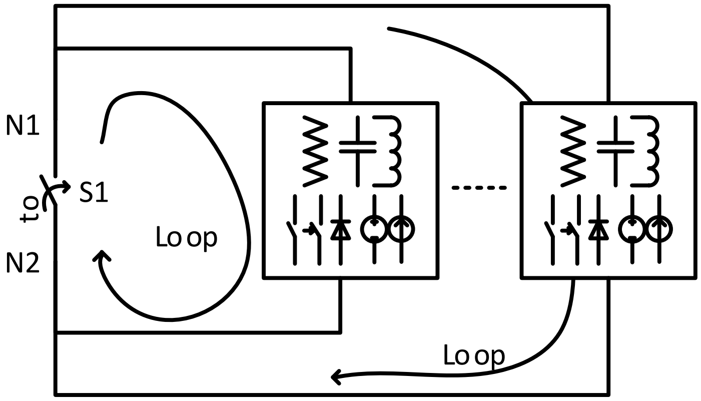


Figure 1.2 The change of a switch state from OFF to ON corresponds to a voltage drop

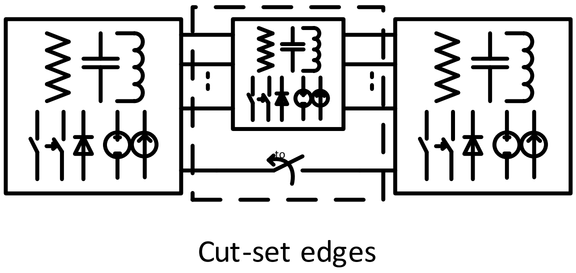


Figure 1.3 The change of a switch state from ON to OFF corresponds to a current drop

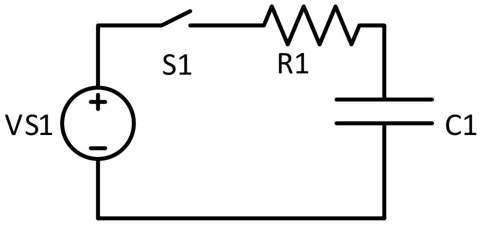


Figure 1.4 Switched active RC network

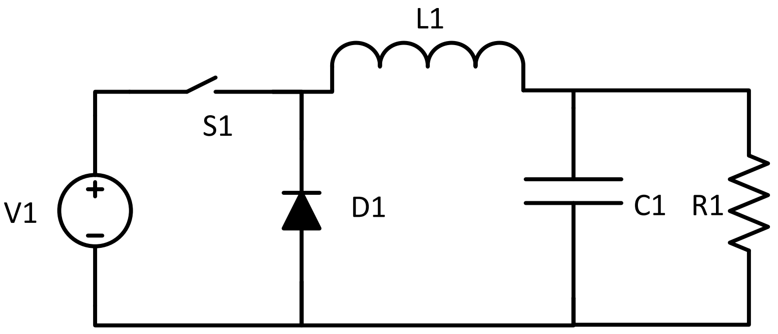


Figure 1.5 DC to DC buck converter

1.3.2 Impulses Are Related to Device Level Behavior

Consider the circuit shown in Figure 1.6. When S1 is OFF, the state space dimensionality is 2 with both capacitor voltages being state variables. However, when S2 is ON, the dimensionality is 1 and only one capacitor voltage can be a state. At a transition from the first case to the second, both capacitor voltages will be discontinuous (assuming the voltages were not identical immediately preceding the switch transition), and a current impulse will flow through the capacitors and switch.

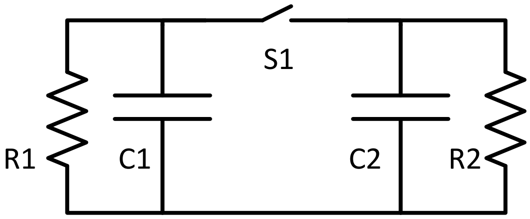


Figure 1.6 Switched passive RC network

This example illustrates the fact that in switched RLCM networks with finite inputs, impulses may arise due to a discontinuity in capacitor voltage or inductor current regardless of the status of the voltage or current as state variable (both before and after the transition). Therefore, when using state space models, it is important not to let the notion of state misguide us to wrong conclusions. In this chapter and in our simulation algorithm, we allow impulses to be legitimate signals and thus we establish independence between the notion of state and the possibility of discontinuity, and thus between the state and its impulsive derivative.

To summarize, independently of any other notions, at the device level, the following is always true:

- For any capacitor whose voltage v_C undergoes a step change, the current $i_C = C\frac{d}{dt}v_C$ will be an impulse.
- For any inductor whose current i_L undergoes a step change, the voltage $v_L = L\frac{d}{dt}\, i_L$ will be an impulse.

1.3.3 Impulses in Switched RLCM Networks

In switched network analysis, admission of impulses is critical for obtaining the correct response.

Impulses arise in the network as a result of interaction between the non-linear i-v characteristic of the ideal switch and the time derivatives in the i-v characteristics of storage elements. Such impulses allow one to model an instantaneous transfer of energy between storage elements. This transfer of energy is a counterpart for the discontinuity in the corresponding variables. In this thesis, we allow impulses in the analysis, eliminating the assumption that the capacitor voltages and inductor currents are continuous, and eliminating the need to analyze discontinuities explicitly. We can characterize a discontinuity by addressing the relationship between the changes caused by switch transitions and the impulses that arise. The relationship can be embedded in the state space model for implicit handling of discontinuities, or a simple (compared to the methods presented in literature) explicit re-initialization equation can be derived.

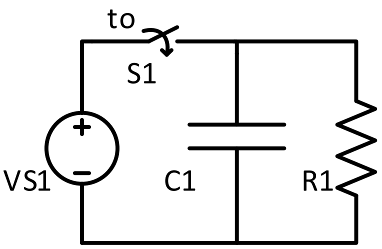
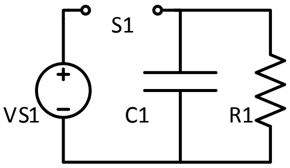
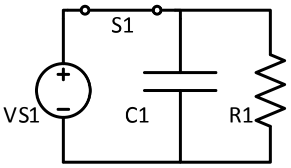
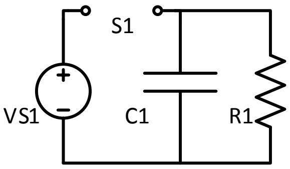
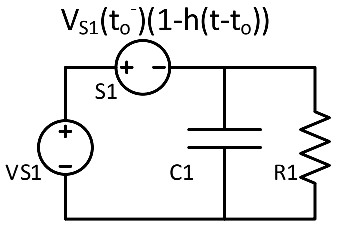
In our simulation algorithm, similar to [4] method, we made use of another type of impulse, which appears in the analysis to help us find correct switching configuration. This type of impulse can help us find the simultaneous solution satisfying RLCM network variables and switch states. That is, some trigger switch transitions can cause disturbances in the network putting a storage element at a critical state. In this critical state, an impulse would normally arise; however, a change in an internally controlled switch state absorbs the disturbance. This type of impulses can be used to trace a sequence of switching events leading to a stable simultaneous solution.

1.4 Contributions

This thesis contributes the following:

- Re-initialization equations are proposed to be part of the ideal switch model and the corresponding state space model thus simplifying the analysis of switched RLCM networks. Unlike the approaches in literature, which consider the re-initialization as an independent step (additional analysis at the instant of switching), the method in this thesis gives a complete characterization of the switched RLCM network at the instant of switching based on the network following the event as shown in Table 1.1.
- A model for ideal switches and a corresponding state space model is introduced as a possible future work to model switched RLCM using a single model.
- An object-oriented application program was designed and implemented to simulate switched RLCM networks using state space modeling and the theory discussed in this thesis.

Table 1.1 A comparison between the approach discussed in this thesis with the approach discussed in literature

<table>
<tr><td colspan="4">
</td></tr>
<tr><th></th><th>t < to</th><th>t = to</th><th>t > to</th></tr>
<tr><td><b>Literature</b></td><td>
</td><td>No circuit equation to map variables</td><td>
</td></tr>
<tr><td><b>Thesis approach</b></td><td>
</td><td colspan="2">
</td></tr>
</table>

1.5 Thesis Organization

In the following chapters of this thesis, the theory that is used to develop the simulation algorithm is discussed first followed by the developed software. In chapter 2, RLCM network modeling, and control system modeling are discussed. The chapter introduces device models, ideal switch models in particular, and system models. Chapter 3 discusses the analysis of the models introduced in chapter 2. It focuses on addressing the analysis challenges of switched RLCM networks. The design and implementation of the simulator are then discussed in chapter 4. In chapter 5, test cases and results are presented. Chapter 6 then introduces an interesting future work based on generalizing the discussion of chapter 2 and 3. Finally, the thesis is concluded in chapter 7.

Chapter 2 Modeling

2.1 Network Modeling

Modeling is prerequisite to simulation, and the choice of model plays an important role in analysis and affects its complexity. In this project, state space models are used for system level modeling. State space models for switched RLCM networks are not unique. An important difference between state space models stems from the choice of device level models. In the following sections, various alternatives for device level models will be discussed. Afterwards, the state space models that we obtain based on these models will be discussed.

In this thesis, the terms “open interval” and “closed interval” will have the specific meanings defined here.

Open interval will refer to a time interval that is free of switch transitions and is open at the beginning and at the end. For such an interval, we assume that we have been given or have computed a consistent initial condition.

Closed interval will refer to a time interval that is free of switch transitions and is closed at the beginning. In this interval, we assume that the consistent conditions satisfying KVL and KCL are available. We call such consistent conditions final conditions[1].

2.1.1 Device level

In this project, we are interested in modeling switches as ideal devices. Such models retain the essential characteristics of the devices but do not include features of secondary importance, such as parasitic effects or stochastic behavior. We focus on switched RLCM networks that are comprised of elements from three groups: RLCM elements, independent sources, and switching elements. There are

---

[1] Consistent conditions associated with a closed interval are called final conditions because they are typically the output of the preceding open/closed interval.

conventional models for devices in the first two groups as summarized in Table 2.1 and Table 2.2. However, many different models for switching elements have been proposed in the literature. While each of those models preserves the characteristics of an ideal switch, different models are defined in this chapter to simplify system level analysis in the next chapter.

Table 2.1 v-i and i-v characteristic of RLCM devices

Device	v-i characteristic	i-v characteristic
Resistor	$v = R\,i$	$i = \frac{1}{R} v$
Conductor	$v = \frac{1}{G}\,i$	$i = G\,v$
Capacitor	$v = v(t_0) + \frac{1}{C}\int_{t_0}^{t} i(s)\,ds$	$i = C\frac{d}{dt}v$
Inductor	$v = L\frac{d}{dt}i$	$i = i(t_0) + \frac{1}{L}\int_{t_0}^{t} v(s)\,ds$
Mutual inductance	$\bar{v} = \bar{\bar{L}}\frac{d}{dt}\bar{\imath}$	$\bar{\imath} = \bar{\imath}(t_0) + \bar{\bar{L}}^{-1}\int_{t_0}^{t} \bar{v}(s)\,ds$

Table 2.2 v-i and i-v characteristic of independent sources

Independent source type	v-i characteristic	i-v characteristic
Voltage source	$v = specified\ input$	$i = determined\ by\ KCL$ $at\ a\ node\ to\ which\ it\ is\ connected$
Current source	$v = determined\ by\ KVL\ of$ $a\ loops\ to\ which\ it\ belongs$	$i = specified\ input$

Switching elements:

Switching elements are at the core of this research. These elements can be in one of two conduction states: conducting (ON) and non-conducting (OFF). The conduction state may be controlled internally or externally. For example, a diode would be represented as an internally controlled switch. As another example, a MOSFET would be represented as an internally controlled switch, if the gating signal is derived from network variables, or as an externally controlled switch, if the gating signal is specified by the user.

Ideal switching elements have the following characteristic: $i = 0$ or $v = 0$. Some alternative models that have an equivalent characteristic are listed in Table 2.3.

Table 2.3 Ideal switch models

Name	ON-State Model	OFF-State Model
Short-Open-Circuit (SOC)	Short circuit	Open Circuit
Zero-Valued Independent Source (ZVIS)	0-V Voltage source	0-A Current source
Step Function Independent Source (SFIS)	Step to 0-V voltage source	Step to 0-A current source

Each model from Table 2.3 brings advantages and disadvantages as we will see later. Conduction state is the only parameter of the first two models. They are very intuitive and the information already presented suffices. The third model depends on additional variables. Table 2.4 gives the details of the model, where $h(.)$ represents the unit step function and t_o is the instant of switch transition.

Table 2.4 Voltage and current characteristic of SFIS

variable	OFF-State Model	ON-State Model
Voltage	$v = v(t_o^-)\left(1 - h(t - t_o)\right)$	$v = 0$
Current	$i = 0$	$i = i(t_o^-)\left(1 - h(t - t_o)\right)$

The importance of this model stems from the fact that it is event dependent. The first two models in Table 2.3, model the ideal switch characteristic based on its conduction state only, independent of the switch condition at the transition instant. This model, on the other hand, includes a derivative at the transition instant to incorporate the switch condition into the model. The addition of the switch condition does not alter the characteristic in that $i = 0$ or $v = 0$ is maintained. However, the derivative of voltage or current may have a non-zero value at t_o; thus the model includes additional information about the transition event.

2.1.2 Network (system) level modeling

To simulate switched RLCM networks, we will be using state space models. The state space model description will vary slightly, yet in an important way, based on the choice of device models. More specifically, the choice of switch model, as will be presented in next chapter, affects the amount of information [about the switched network] the resulting state space model includes. In this section, the models will be introduced, and the analysis details will follow in the next chapter.

Switched Network Incomplete State Space Model[2]:

In this model, we represent switches as Short-or-Open-Circuits (SOC). This representation is the simplest, and its inclusion in the state space model derivation can be handled in the obvious way. Comparing this state space model to the ones that will follow, this model is the simplest because switch variables do not appear in the model.

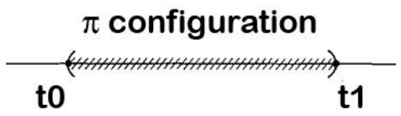
A single state space model is not sufficient to describe a switched RLCM network. Instead, a state space model is developed for each network configuration, where the configuration is identified by state of the switches. Each of these models is valid in an open interval between two switch transitions. This model requires that consistent initial conditions be provided at the beginning of each interval.

---

[2] Incomplete State Space Model for short.

Table 2.5 summarizes this model for a given switch configuration, π.

Table 2.5 Summary of Switched Network Incomplete State Space Model

	Summary	
Equation form	State equation	$M_\pi \, \dot{x} = A_\pi \, x + B_\pi \, u + B_{\pi,1} \, \dot{u}$
	Output equation	$y = C_\pi \, x + D_\pi \, u + D_{\pi,1} \, \dot{u}$
Variables	x	Vector of state variables
	$\dot{x}$	Vector of first derivatives of state variables
	u	Vector of input variables
	$\dot{u}$	Vector of first derivatives of input variables
	y	Vector of outputs variables
Validity interval		
Validity requirements	Consistent $x(t_o^+)$ must be provided.	

The incomplete model can also be obtained by modeling switches as Zero-Valued Independent Sources (ZVIS). The only difference between the ones based on SOC models, and the ones based on ZVIS is the definition of state space model variable u. This is summarized in Table 2.6.

Table 2.6 u variable as defined by SOC and ZVIS based state space models

variable	SOC based	ZVIS based
$\boldsymbol{u}$	Input vector	Input vector concatenated with zero entries for switches
$\dot{\boldsymbol{u}}$	Vector of first derivatives of input variables	Vector of first derivatives of input variables concatenated with zero entries for switches

Switched Network Complete State Space Model[3]*:*

This state space model is based on representing the switches by Step Function Independent Source (SFIS) models. This state space model is a generalization of the incomplete state space model where switches are represented by (ZVIS) models. In this state space model, however, non-zero valued derivatives of switch variables are included in the $\dot{u}$ variable. Including the derivatives extends the interval for which the model is valid to include the switching instant, which makes the validity requirement a trivial one. Table 2.7 summarizes this model for a given switch configuration, π.

---

[3] Complete State Space Model for short.

Table 2.7 Summary of Switched Network Complete State Space Model

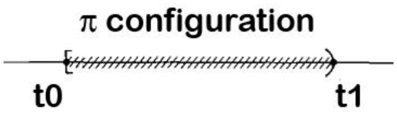
	Summary	
Equation form	State equation	$M_\pi \, \dot{x} = A_\pi \, x + B_\pi \, u + B_{\pi,1} \, \dot{u}$
	Output equation	$y = C_\pi \, x + D_\pi \, u + D_{\pi,1} \, \dot{u}$
Variables	x	Vector of state variables
	$\dot{x}$	Vector of first derivatives of state variables
	u	For independent sources, u_i is input
		Turning ON switches, $u_i = v(t_o^-)\,\big(1 - u(t - t_o)\big) = 0$
		Turning OFF switches, $u_i = i(t_o^-)\,\big(1 - u(t - t_o)\big) = 0$
	$\dot{u}$	For independent sources, $\dot{u}_i$ is first derivative of input
		Turning ON switches, $\dot{u}_i = \frac{d}{dt}\Big(v(t_o^-)\,\big(1 - u(t - t_o)\big)\Big) = -v(t_o^-)\,\delta(t - t_o)$
		Turning OFF switches, $\dot{u}_i = \frac{d}{dt}\Big(i(t_o^-)\,\big(1 - u(t - t_o)\big)\Big) = -i(t_o^-)\,\delta(t - t_o)$
	y	Outputs vector
Validity interval		
Validity requirements	Consistent $x(t_o^-)$ must be provided	
	For switches turning ON, consistent $v(t_o^-)$ must be provided	
	For switches turning OFF, consistent $i(t_o^-)$ must be provided	

Table 2.7 indicates that this model has more validity requirements than the earlier model. Note, however, that all the required variables are readily available in simulation applications. After discussing the analysis of these state space models in more details, a method to eliminate, or deduce, the consistent switch variable requirements will be presented for added generality.

State Space Model Formulation:

State space model formulation is an important aspect of the analysis. Because this thesis is aimed at developing a computer aided simulation application, the formulation algorithm must be well defined and suitable for computer environments. The method we chose is discussed in [7].

While the method presented only discusses the RLCM elements, we will also be adding switches. Switches modeled as Short-or-Open Circuits are trivial to add. We include switches modeled as independent sources similar to actual independent sources. We can incorporate switches in the general formulation algorithm by using the following tree edge weighting strategy:

$$w(VS) < w(ON\ switch) < w(c) < w(R) < w(L) < w(OFF\ switch) < w(CS)$$

2.2 Control System Modeling

Control system model describes the switch state control of internally controlled switches. The control system, in our analysis and simulation, will be constructed such that it is an independent entity from the network model. While this choice might not be the most concise, it allows for generality which is one of the project requirements.

2.2.1 Device Level

The control system consists of functional blocks, signal sources, and signal sinks. The functional blocks are defined by their inputs, [states,] outputs, and a function that operates on the inputs [and the states]. The signal sources and signal sinks, as their name suggest, source the control system with signals and sink signals by controlling switch states. All of the blocks operate in the ideal way such that the output is deterministic given the block definition.

Functional Blocks:

Functional blocks are divided into two categories; dynamic, and non-dynamic. The dynamic blocks depend on a state or states and present inputs. On the other hand, non-dynamic blocks depend on present inputs only.

Dynamic Functional Blocks

From this category, we are interested in integrators only. Integrators are defined by a single input, a state, and output. This block is defined as in Table 2.8.

Table 2.8 Definition of dynamic function blocks

Inputs	u
Outputs	y
State	x
Characteristic function	$y = \int u(t)\,dt$
Numerical integration model	$x(t+dt) = x(t) + u\,dt$ $y(t) = x(t)$

Non-Dynamic Functional Blocks

This category consists of adders, comparators, gain blocks, and multipliers. The characteristic of each of these blocks is dependent on present input to the block only. Table 2.9 summarizes these blocks. In Table 2.9, u_i is the i^{th} input to the block, and y is its output.

Table 2.9 Output characteristics for non-dynamic function blocks

	Characteristic function
Adder	$y = u_1 + u_2$
Comparator	$y = \begin{cases} y_{true}, & u_1 > u_2 \\ y_{false}, & u_1 \leq u_2 \end{cases}$ Where y_{true}, and y_{false} are parameters
Gain block	$y = g\, u_1$ Where g is a parameter
Multipliers	$y = u_1\, u_2$

Signal Sources:

Signal sources blocks input signals to the control system. A subset of these blocks forms the input interface between the electrical network and the control system. Based on the input to these blocks, we define two categories: sensors, and signal generators. Signal generators are intended to be independent blocks generating signals directly to the control system. On the other hand, sensors are blocks attached to the electrical network to allow the transfer of signals to the control system. Table 2.10 summarizes the two signal source block types.

Table 2.10 Parameters and characteristics of signal sources

	parameters	Characteristic function
Sensors	The device to be considered, d	$y = \begin{cases} v\ of\ device\ d, & v\ sensor \\ i\ of\ device\ d, & i\ sensor \end{cases}$
	The variable to be measured, i or v	
Signal Generators	Function F	$y = F(\varphi)$
	Parameter set φ independent to electrical network	

Signal Sinks:

Signal sinks form the output interface between the control system and the electrical network. This set consists of digital sinks. The digital sinks are switch controllers. The state of switches a sink controls equals the binary bit-value of the input to the block as shown in Table 2.11.

Table 2.11 Definition of signal sinks

Input	u
Parameter	Threshold
Output	$y = \begin{cases} ON, & u < threshold \\ OFF, & u \geq threshold \end{cases}$
Switch interface	Switch state = y

2.2.2 System level

Unlike the system level model of the electrical network, this control model does not formulate a single system model. In the control system, we assume that the variables do not have circular dependences. This assumption makes it possible to find a sequence of function invocations, starting with system inputs (from signal sources) to system outputs (to signal sinks). To obtain the outputs of the system, we evaluate the characteristic function of each block on its inputs in the order specified by the generated function invocations sequence. The assumption that variables do not form circular dependency does limit the set of control systems that can be simulated. However, such systems are not within the scope of our interest.

Chapter 3 Analysis

3.1 Analysis of Switched Networks

In this chapter, the state space models presented in Chapter 2 will be discussed in more detail with a focus on switch transition instants. To study the state space models in isolation from the dependencies of the control variables of switches, networks with only externally controlled switches will be considered initially. Networks containing internally controlled switches will be considered later in the chapter.

3.2 Incomplete State Space Model Analysis (Switches Modeled as SOC)

In the analysis of networks containing ideal switches, the Short-Open Circuit model for switches is very common. While this model describes the network, given the states of ideal switches, in a very intuitive manner, handling switch transitions is complicated. Despite the fact that the model eliminates the switches nonlinearity in the open intervals where the switches states are constant, it does not properly address the correlation between network variables in successive time intervals. The inability of this model to describe the switching events necessitates an additional analysis to transfer final conditions to initial conditions. This additional analysis fills the gap between two successive time intervals described by two incomplete state space models.

This approach to analyzing a switched network ignores switch variables locally, in that each individual state space model is independent with respect to switch variables. This makes the task of transferring final conditions to initial conditions for storage element variables non-trivial. Considering the network model in any of the open intervals where switches are replaced with their Short-Open Circuit model, it is not obvious what changes initiated and caused step changes in the variables associated with the storage elements .

This problem has been addressed in the literature by developing explicit equations for transfer of conditions. [4] relies on a special two-step integration algorithm based on Laplace transform to perform the transfer. Their method, however, was designed to accompany nodal analysis and companion modeling techniques. A more recent work by [5] developed an explicit transfer of condition equations for state space models. Formulation of their equations, however, requires generating a Moore-Penrose pseudo inverse and finding the null space of a matrix for every topology of interest.

Finding consistent initial conditions after switch transitions is not sufficient for simulation. The combination of incomplete state space models and the re-initialization equations do not describe a switched RLCM network. Instead, they describe a sequence of linear RLCM networks. This is because, at the switch transition instant, the combination describes the step changes, if any, in energy storage element variables. The combination does not describe the remainder of the RLCM network. For this reason, at least one more analysis must be performed to identify the presence of impulses through or across switches. This task cannot be performed simply by substituting impulses for state variables in the equations that we already have. This is because the impulses, of capacitor currents and inductor voltages, we computed from the re-initialization equations are present at t_o, or equivalently during (t_o^-, t_o^+). However, we do not have any equation that describes the switched RLCM network during (t_o^-, t_o^+).

A section that will follow addresses the reason why the additional analyses are required. The problem is then tackled by extending the ideal switch characteristic leading to a more expressive switch model.

3.3 Incomplete State Space Model Analysis (Switches Modeled as ZVIS)

This model is tempting to use especially for computer-aided analysis algorithms as it is well known that a 0-V voltage source is equivalent to a short circuit, and a 0-A current source is equivalent to an open circuit. However, the network analysis using this model is prone to the same issues that arise when modeling switches as SOC.

3.4 Completing the State Space Model

Before delving into details, we will state explicitly why consistent final conditions are not necessarily consistent initial conditions. Consistent final conditions, by definition, combined with inputs and switch configuration before switch transitions, satisfy both KVL and KCL. However, after switch transitions, the change in network topology makes it possible that the zero-voltage or zero-current property around a loop or through a cut-set is violated by some of the final conditions. To account for this violation, the re-initialization equations, which give the consistent initial conditions, are used. Figure 3.1 and Figure 3.2 illustrates KVL and KCL during a switch transition.

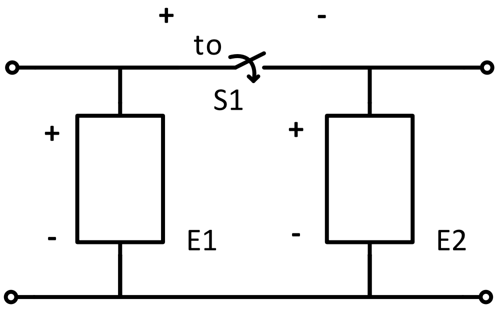


Figure 3.1: KVL must be conserved for all time t: v(E1, t) - v(E2, t) - v(S1, t) = 0

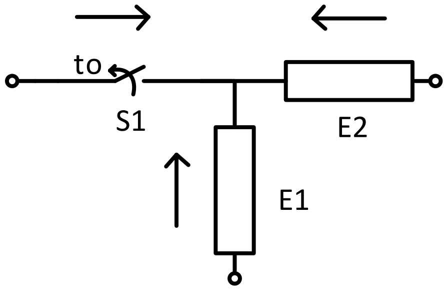


Figure 3.2: KCL must be conserved for all time t: i(E1, t) + i(E2, t) + i(S1, t) = 0

The combination of incomplete state space models and the re-initialization equations are not sufficient to characterize the switched RLCM network. In this combination, other than the re-initialization equations, which map consistent final to initial conditions, the steps in capacitor voltages or inductor currents make no sense. Any attempt to reference these voltage or current steps outside the scope of the re-initialization equations leads to violation of the first derivative of KVL or the first derivative of KCL. Because state space models are constructed assuming KVL and KCL, the incomplete state space models are invalid outside the open interval. It is also why we must perform another independent analysis in order to find impulses around the switched RLCM network at the instant of switching. Table 3.1 and Table 3.2 illustrates the violation of the derivatives of KVL or KCL when erroneously attempting to use any of the incomplete state space at the instant of switching.

Table 3.1 Erroneous use of network model at turning ON switching instant with switches modeled as SOC

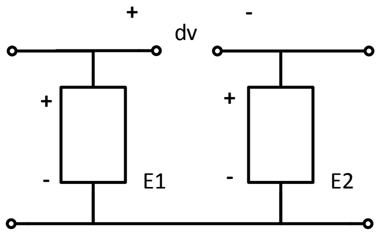
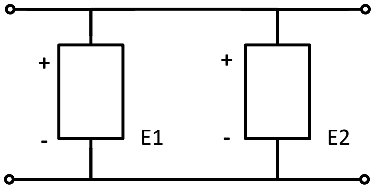
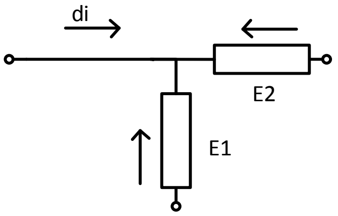
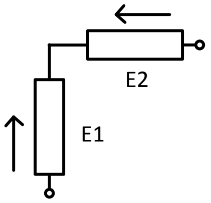
	t < t0	t > t0
Equivalent network		
KVL	No requirements	v(E1, t) - v(E2, t) = 0
Inconsistency	d/dt[v(E1, t0) - v(E2, t0)} != 0	
Correction	d/dt[v(E1, t0) - v(E2, t0)} != - dv δ(0)	

Table 3.2 Erroneous use of network model at turning OFF switching instant with switches modeled as SOC

	t < t0	t > t0
Equivalent network		
KVL	i(E1, t) + i(E2, t) + i(S1, t) = 0	i(E1, t) + i(E2, t) = 0
Inconsistency	d/dt[i(E1, t0) + i(E2, t0)} != 0	
Correction	d/dt[i(E1, t0) + i(E2, t0)} = di δ(0)	

The switch model used to construct incomplete state space models misleads us to ignore the voltage derivative around, or the current derivative through a switch. Once this piece of information is recovered, we can easily verify that loop voltage derivative or the cut-set current derivative is equal zero at all times. And thus we can relate the non-zero derivative of storage element's variables with the non-zero derivative of switches variables. The switches force the voltage around themselves or the current through them to zero. Using this property simplifies the analysis and extends the validity interval of our model. So, preserving the relevant derivatives, KVL and KCL can be maintained even during the instant of a switch transition. Thus the state space model after the transition can be made valid at the transition instant.

After addressing the importance of switch derivatives in the analysis, it is reasonable to generalize, or complete, the conventional switch characteristic from chapter 2. In the updated characteristic, we will include the switch derivative, which we require that the switch model satisfies. Table 3.3 summarizes the conventional and the complete switch characteristic.

Table 3.3 Complete ideal switch characteristic

	ON-State	**OFF-state**
Conventional characteristic	$v = 0$	$i = 0$
Complete characteristic	$v = 0$ and $\frac{d}{dt}v = -v(t_o^-)\delta(t - t_o)$	$i = 0$ and $\frac{d}{dt}i = -i(t_o^-)\delta(t - t_o)$

Completing the state space model, thus, means that the additional analysis methods are eliminated. The state space model itself can handle any discontinuities. A complete state space model describes the network at the instant of switching, or equivalently, the model contains the information necessary to transfer the final conditions into consistent initial conditions. Moreover, having a network and a corresponding state space model that is valid at the switch transition instant makes the task of finding impulses around the switched RLCM network trivial.

3.5 Complete State Space Model (Switches Modeled as SFIS)

In the complete state space model presented in Chapter 2, we use the SFIS switch model, which is consistent with the complete switch characteristic presented in Table 3.3. It preserves enough information to fill the gap between final and initial conditions, while maintaining the simplicity of an incomplete state space model. The complete state space model specifies that given the switch configuration, we let ON switches be step-to-zero voltage sources, and OFF switches be step-to-zero current sources. The step size of each of the step sources is equal to the switch variable specified by the final conditions, $v_{switch}(t_o^-)$ or $i_{switch}(t_o^-)$. In simulation applications, we normally know switch states at the least, in which case we can always evaluate the step size using the state space model prior to the switch transitions. If otherwise is the case, methods in a section that will follow address the problem.

Based on this method, switches are necessary to correctly redistribute energy in discontinuity conditions. Derivatives of switch variables take care of state re-initialization implicitly in this model. In some networks, it is possible that inconsistent conditions appear, usually defined by user, in loops or cut-sets that do not contain any switches. In this case, consistency is assumed, and one or more of the final conditions is neglected. Such circuits can be identified easily, and an equivalent circuit can be constructed to eliminate the limitation.

In the tree/co-tree representation of the network, a capacitor failing to be a branch or an inductor failing to be a link indicates dependence of the violating element on another that succeeded to be. Therefore, if there exists a tree such that all voltage sources and all capacitors are branches, and all current sources and all inductors are links, irrespective of where switches go, then either all energy storing elements are independent or a switch exist in the same loop or cut-set. Such a test can be done by choosing the weight function for edges properly. Networks that fail this test, potentially, exhibit the problem described above. To solve the problem, an equivalent network must be constructed by adding ON switches in series with capacitors failing the test, and OFF switches in parallel with inductors failing the test.

3.6 Explicit State Re-initializing[4]

The complete state space model is capable of handling inconsistent initial condition, by relying on consistent final conditions only. However, it is common that we are given a network with final conditions not completely specified; switch states were not given. In this case, the method based on the complete state space model will experience difficulty because all the analyses discussed previously were based on the assumption that we started with given final conditions.

To handle this case, we have two choices, explicitly reinitialize energy storage variables and use the complete state space model starting with initial conditions, or find consistent switch states to complete specifying final conditions. Both approaches will be considered. A simple, as compared to the literature, explicit re-initialization equation will be derived first, followed by a method to find consistent switch states.

Given the state equation

$$M\frac{d}{dt}x = Ax + Bu + B_1\frac{d}{dt}u$$

The equation above is the general form of a complete state space model. If switch states are not specified as part of the final conditions, then u and $\dot{u}$ which depend on switch variables are unknowns. Thus the state equation is not completely specified. In the following analysis, we will attempt to derive explicit re-initialization equation independent of switch variables based on the state equation.

---

[4] This section uses same matrix/vector naming notation from [7].

We can derive the re-initialization equation as follows:

$$x(t_o^+) = x(t_o^-) + \int_{t_0^-}^{t_0^+} \frac{d}{dt} x \, dt$$

$$x(t_o^+) = x(t_o^-) + \int_{t_o^-}^{t_o^+} M^{-1} \left(Ax + Bu + B_1 \frac{d}{dt} u \right) dt$$

$$x(t_o^+) = x(t_o^-) + M^{-1} \, B_1 \int_{t_o^-}^{t_o^+} \frac{d}{dt} u \, dt$$

The integral is non-zero only when the derivative is impulsive, and the equation becomes

$$x(t_o^+) = x(t_o^-) + M^{-1} \, B_1 \big(u(t_o^+) - u(t_o^-) \big)$$

This equation appeared implicitly in the complete state space model, which allowed us to relate the final and the initial conditions. However, if the initial switch states at t_o^- are unknown, then so is $u(t_o^-)$. To work around this issue, notice the following set of equations. From the fundamental cut-set equations, we can derive the following equation:

$$\begin{bmatrix} F_{14}^t & 0 \\ 0 & F_{41} \end{bmatrix} u = \begin{bmatrix} F_{24}^t - I & 0 & 0 \\ 0 & 0 & -I - F_{42} \end{bmatrix} \begin{pmatrix} v_c \\ i_l \end{pmatrix}$$

where $u = \begin{pmatrix} v_{vs} \\ i_{cs} \end{pmatrix}$ for all devices modeled as independent sources (including SFIS switches), and v_c and i_l are vectors of all capacitor voltages and all inductor currents respectively.

By inspecting the B_1 matrix, we notice that $\begin{bmatrix} F_{14}^t & 0 \\ 0 & F_{41} \end{bmatrix}$ is a right factor, thus let

$$K = \begin{bmatrix} F_{14}^t & 0 \\ 0 & F_{41} \end{bmatrix}$$

and let

$$B_1 = \hat{B}_1 K$$

Then we can rewrite our re-initialization equation as

$$x(t_o^+) = x(t_o^-) + M^{-1} \hat{B}_1 \big(Ku(t_o^+) - Ku(t_o^-)\big)$$

by using the equation that links initial states with inputs, we obtain

$$x(t_o^+) = x(t_o^-) + M^{-1} \hat{B}_1 \left(Ku(t_o^+) - \begin{bmatrix} F_{24}^t - I & 0 & 0 \\ 0 & 0 & -I - F_{42} \end{bmatrix} \begin{pmatrix} v_{vs}(t_o^-) \\ i_{cs}(t_o^-) \end{pmatrix}\right)$$

where $u(t_o^+)$ is a vector of independent sources and zero entries for all switch variables.

This last equation allows us to reinitialize the state (find consistent initial conditions) given the incompletely specified final conditions and inputs. Notice that we are relying on the derivative of u for re-initialization. In other words, this formulation assumes that there exists a switch configuration such that all capacitor voltages and all inductor currents are states (independent)[5]. Otherwise the initial conditions of some elements will be ignored. This limitation can be eliminated using the same approach discussed in Section 3.5.

This last equation reveals an important fact about the necessity of knowing switch initial states at t_o^-. That is, since the equation does not depend on input values from t_o^- and thus does not depend on switch initial states, then we can opt to initialize our switches to any configuration such that at t_o^-, KVL, KCL, and energy storage element initial conditions are satisfied. This last statement can help us in cases where it is desired to have a fully specified final condition rather than attempting to find an explicit consistent initial condition. Also notice that because we will need the derivative of the switch variables,

---

[5] This is an existence statement. It is not a requirement that the desired switch configuration satisfy this condition.

by same argument, we can simply solve for the voltages and currents of switches at t_o^- using any method we prefer, given we have chosen consistent switch states at t_o^-.

3.7 Network analysis in the presence of internally controlled switches and diodes

Internally controlled switch and diode variables are coupled with other network variables. Therefore, to include internally controlled switches in the network analysis, we must solve for all the variables of the switched network simultaneously. The simultaneous solution must consider the RLCM variables, KVL and KCL, switches control variables, and inputs. That is, it must find switch states, and accordingly switch voltages or currents, and properly deals with discontinuities in energy storage variables.

Attempts to find the simultaneous solution are rarely discussed in the literature. One interesting attempt was proposed by [6]. Their work assumes capacitor voltages and inductor currents are constant at the instant of switching. While the work is interesting, we desire to develop a simulator that is capable of analyzing any switched RLCM network without imposing any such constraints.

A consistent simultaneous solution can be found iteratively, without imposing continuity constraints on capacitor voltages and inductor currents. There are several valid iterative approaches, one of which is evaluating all possibilities to find the consistent ones (such as bond graph method). However, since the number of possibilities grows exponentially with the number of switches, this approach is not feasible. To overcome the exponential growth, we can iterate over topologies according to the intermediate analytical impulses in the network. That is, assume switch control is independent of network variables. Freeze time and final conditions, solve the network, and change switch states according to this new network solution. Keep iterating until all switches control variables are stable.

1. Freeze time at t_o, and freeze $\begin{pmatrix} v_{vs}(t_0^-) \\ i_{cs}(t_0^-) \end{pmatrix}$, and $u(t_0^-)$
2. Until switch states are stable
    a. Identify and update switches that must take a transition
    b. Use $M_\pi\,\dot{x} = A_\pi\,x + B_\pi\,u + B_{\pi,1}\,\dot{u}$ and $y = C_\pi\,x + D_\pi\,u + D_{\pi,1}\,\dot{u}$ of the new topology to evaluate switch control variables at t_o^+. In this step, use final conditions from step 1.

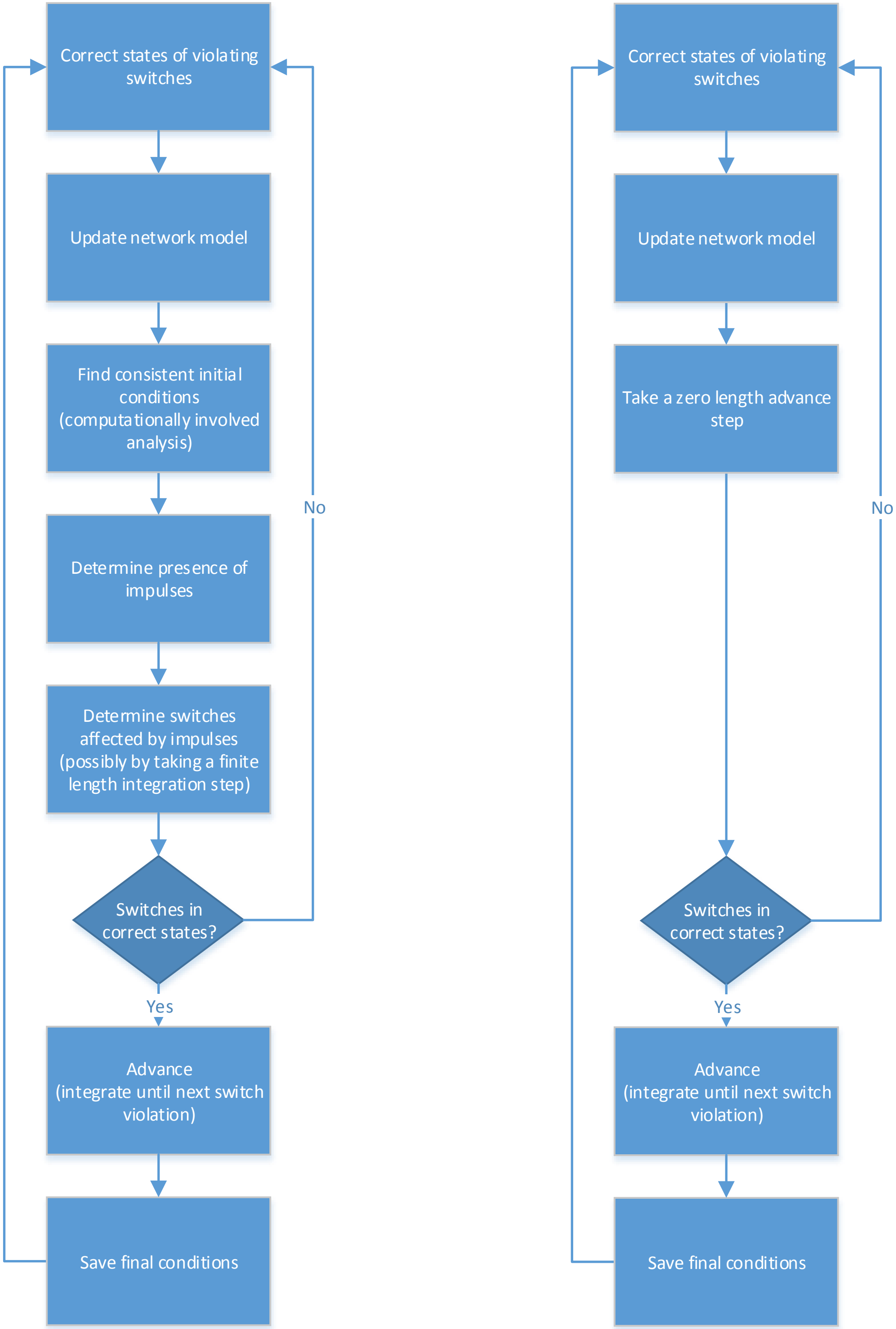
Step 2.b can be performed numerically as follows:

1. Numerically integrate x using $M_\pi\,\dot{x} = A_\pi\,x + B_\pi\,u + B_{\pi,1}\,\dot{u}$ from t_0^- to t_0^+. The integration is equivalent to the explicit re-initialization: $x(t_0^+) = x(t_0^-) + M^{-1}\,B_1\big(u(t_0^+) - u(t_0^-)\big)$.
2. Evaluate variables of interest (needed for control) using $y = C_\pi\,x(t_0^+) + D_\pi\,u(t_0^+) + D_{\pi,1}\,\dot{u}(t_o)$.

The steps to evaluate control variables, given by the algorithm above, can be implemented to be automatic using the same infrastructure used to simulate the state space model in the open interval. However, because we are interested in a fixed time instant, we only make zero length integration steps. The implementation of this algorithm is simple.

3.8 Design of the simulation algorithm

Computer aided analysis and simulation require generalizable algorithms. Since we have a well-defined algorithm for constructing state space models, simulation based on the sequence of complete state space models methods is suitable. Each state space model is solved using numerical integration algorithms. At switch transition instants, the simultaneous solution for control equations and state space model equations is found iteratively, as in the earlier section. Figure 3.3 gives an overview of the simulation algorithm, and compares it with the common algorithm used in literature. Other than the computational simplicity of this method, this method abstracts the complexities of switch behavior during the search for correct switch states.


Correct states of violating switches
Update network model
Find consistent initial conditions (computationally involved analysis)
Determine presence of impulses
Determine switches affected by impulses (possibly by taking a finite length integration step)
Switches in correct states?
No
Yes
Advance (integrate until next switch violation)
Save final conditions
Correct states of violating switches
Update network model
Take a zero length advance step
Switches in correct states?
No
Yes
Advance (integrate until next switch violation)
Save final conditions

(a) Literature **(b) This thesis**

Figure 3.3 Simulation algorithm commonly used in literature as compared to algorithm in this thesis

Chapter 4 Software Design

4.1 Switched RLCM Network Simulator

This chapter provides an overview of the design and implementation of a switched network simulator application program that is based on the theory introduced in the previous chapters. The program was designed to provide fully automatic simulation of a general set of switched RLCM networks. That is, given a “net list“ description of the network by an end user, the program determines the transient response of the network without resorting to end user intervention or consultation to resolve the configuration of the network (switch conduction states). The program was designed using object oriented programming techniques, and was implemented using the C++ programming language and the Interconnected System Modeling Framework (ISMF).

4.2 Use Case

The program expects two input files to be provided by the end user: an interconnected system model or ‘.ism’ file that describes the parameterization and interconnection of components in the network and a scenario or ‘.scn’ file comprised of commands that schedule the sampling of component variables, the modification of component parameters/status, and the end time for the simulation. From these inputs, the program determines the transient response of the network, and produces time series data or ‘.tsd’ files and several log files.

Simulation speed is important for usability of the software. The speed was addressed at the algorithm level; modeling and analysis. In particular, fixed time step integration is sufficient for analyzing ideal switch models, eliminating the need to spend more time at switch transition instants. Moreover, at switch transition instants, the use of complete state space model ensures efficiency.

4.3 Program Architecture

The architecture of the program is highly influenced by the ISMF, which supports the creation of application programs that combine component models and analysis methods. From a functional decomposition perspective, the design considers the following three modules: an RLCM Network Simulator (RLCM-NS), a Switched Network Simulation Extension (SNSE), and a Switch Control System (SCS). The three modules form the core of the simulation engine supported by the ISMF.

4.3.1 Interconnected System Modeling Framework (ISMF)

The ISMF extends the standard Object Oriented Programming (OOP) paradigm to make it convenient to model components and their interconnection and interaction. The framework provides functionality for serialization of objects and for controlling program and data flow.

The framework defines specifications for component models and analysis methods. It uses the first to guide interpreting the '.ism' file to populate component model data (parameters) and to interconnect component models. In addition, the framework can extract component model data (variables) to display or record time series data. The analysis methods are used by the framework to support complex analysis beyond what is implemented by the individual component models.

The framework is more complex than what was addressed in this section. However, because the ISMF is not the main subject of this thesis, the more details will be addressed where necessary.

4.3.2 Analysis Methods and Their Utility in The Context of Functional Decomposition

In this section, the simulator will be examined from an object oriented design perspective. Analysis methods and the classes they rely on will be discussed in the context of the functional decomposition. Component model classes will be presented afterwards. In the discussion of the individual classes, references to the functional decomposition terminology will be made to justify specific design decisions.

The RLCM Network Simulator module (RLCM-NS) combined with the capabilities provided by the `SwitchedRLCMNetwork` class (an InterconnectedDynamicalSystemModel class) is a standalone network simulator. The `StateSpaceModelAnalyzer` class, an analysis method, is the primary analysis tool for RLCM networks in this project. As the name suggests, the analyzer uses state space modeling techniques, and is responsible for all related tasks. Switch state management is abstract and hidden from this part of the simulation process. However, the analyzer was designed to be flexible in order to support the Switched Network Simulation Extension (SNSE).

The SNSE adds the switch transitions simulation capability. The most critical analysis this extension supports is provided by the `DiscontinuityAnalyzer` class, an analysis method. It coordinates with other classes in order to carry the ultimate goal of simulating *switched RLCM networks*. More importantly, it represents switch transitions as input discontinuities, which can be handled by the RLCM-NS.

The Switch Control System (SCS) evaluates switch control states as defined by the end user. It relies on a set of component models, which represent control blocks, to evaluate the control outputs. The control system is passive in the sense that its outputs are calculated and made available for other modules to reference. In particular, the SNSE decides when and how to use the control system outputs. Therefore, it is the responsibility of the SNSE to ensure consistency and correctness between actual switch states and the corresponding control system outputs. The `SwitchedRLCMNetwork` class, supported by the ISMF, uses the control variables to identify switch transition free time intervals where pure RLCM network simulation techniques are sufficient. The SNSE uses the control outputs to find consistent switch states.

RLCM Network Simulator Module

The RLCM network simulation module consists of the `StateSpaceModelAnalyzer`, and the component models it supports or interacts with.

This module supports the basic RLCM elements represented by passive elements, and independent sources. The `StateSpaceModelAnalyzer` considers switch configuration as is, with partial support to `SwitchableElement` component models, the hierarchy of different types of switches, and analyzes the equivalent RLCM network using state space model techniques.

The `StateSpaceModelAnalyzer` is responsible for all tasks related to state space modeling and analysis. It is responsible for model generation, maintenance, and use. The state space model for an RLCM network defines a state equation and output equations. These equations depend on the state vector, the input vector, and the input derivative vector, and they provide the state derivative vector and network output vector. This analysis method takes care of extracting, evaluating, and storing these variables locally.

This simulation module works in the context of switched RLCM network simulation, requiring certain degree of flexibility. Because switch transitions represent discontinuities, the `StateSpaceModelAnalyzer` supports impulses: it handles both continuous time and time-as-a-limit when evaluating the state space model equations. Moreover, because the state space of a switched RLCM network is time varying, this analysis method uses dynamic vector representations for the state vector and its derivative.

Figure 4.1 gives an overview to the most important classes in this module.

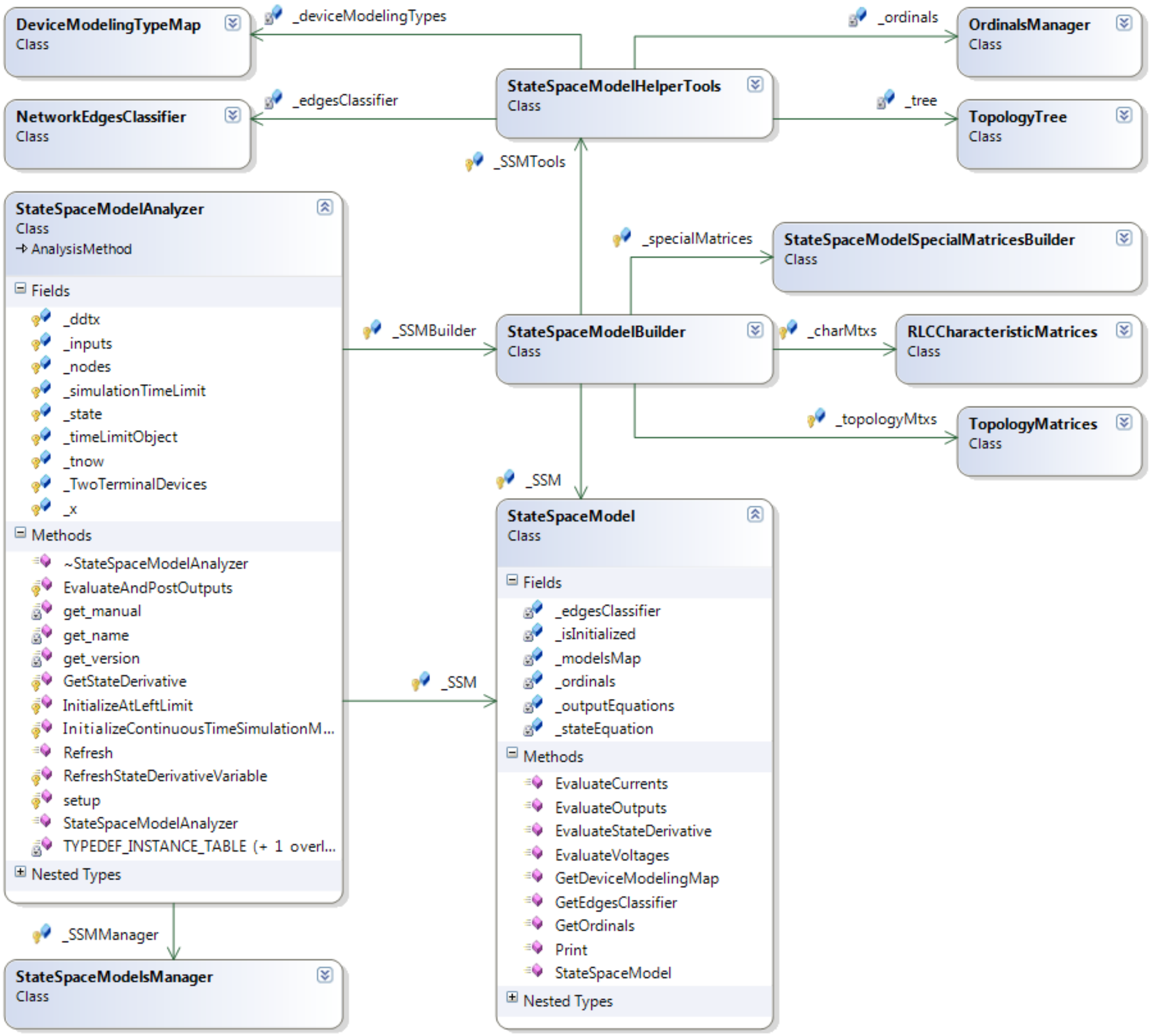


Figure 4.1 Class diagram for State Space Analyzer and the classes it is composed of

Analysis Method: State Space Model Analyzer Class

The `StateSpaceModelAnalyzer` is the core analysis method for the RLCM-NS. It models and analyzes the RLCM networks using state space model techniques. Because this analysis method is only concerned about the RLCM elements, it uses switch states as a fact (except at initialization when it ignores their states). Given switches states, they are included in the state space model as an equivalent element from the RLCM network set, namely voltage sources and current sources. Modeling switches as independent sources is not the responsibility of this analysis method. Instead, `SwitchableElement` is designed to handle the most complex operations, leaving the `StateSpaceModelAnalyzer` with little to do. Modeling switches as independent sources is not always doable due to possible violation of assumptions of the state space model construction algorithm. This analysis method addresses these conditions by opting to model violating switches as resistors with appropriate resistance values.

The `StateSpaceModelAnalyzer` is required to manage multiple topologies for any given network. This requirement affects the design of the state space model generation process, and the design of `StateSpaceModel` elements.

Because this project assumes generality, and because networks often return to particular topologies several times during simulation, it is desirable to avoid reconstructing the `StateSpaceModel`s for such topologies. Thus, the `StateSpaceModel`s are stored and accessed by their corresponding topology. This additional feature requires careful design. The information about how a state space model was constructed is essential for interpretation and usage. This makes it important to identify such information, and store a deep copy per topology that is guaranteed not to be affected by future executions of the construction methods.

The elements of `StateSpaceModel`s needed to be designed carefully as well. The `StateSpaceModelAnalyzer` class must not be concerned about the complexities of the SNSE. However, it has to support simulating multiple RLCM networks per switched network. Because there is only a single RLCM network active at a time, the state space model variables were designed to be dynamic. The state vector and the state derivative vector, unlike normal vector objects, have dynamic size. The size of dynamic vectors expands or shrinks to fit the dimensionality of the active state space model. In some sense, the change in vector size theoretically corresponds to changes in the state space.

Dynamic vectors do not solve all the problems related to the change of dimensionality of the model. When the state space changes, the interpretation of state vector changes too. This means that the ordering of elements in the vector must change, some elements may be removed, and new elements may be added. Changes happen at boundaries of piecewise linear time intervals, and after the `DiscontinuityAnalyzer` executed `FinalizePiecewiseLinearStep`, which ensures `TwoTerminalDevice` objects have reset their final conditions. The state space changes are therefore managed by re-extracting final conditions from `Capacitor` and `Inductor` objects.

State Space Model Builder Class

The `StateSpaceModelBuilder` is the most complex part of the `StateSpaceModelAnalyzer`. It implements the algorithm presented by [7] to construct the state equations, and uses the same infrastructure to construct output equations for all network variables. To support the requirements imposed by the `StateSpaceModelAnalyzer`, this object also prepares safe-to-copy information that is necessary to interpret and use the `StateSpaceModel`.

At initialization, typically at t = 0, the end user is not required to give the final switch states as part of the final condition specifications. The `StateSpaceModelBuilder` is required to support the initialization process, as discussed in the earlier chapters. This object therefore supports two state space model generation modes; initialization and normal. The initialization mode assumes final switch states were not give, and constructs a state space model that can be used to recover this information. The normal mode uses switch states as a given to construct the state space model.

State Space Model Class

A `StateSpaceModel` class contains the state equation, output equations, and all the information needed to use it and interpret its results.

The state space model equations are implemented as `StateSpaceModelEquation` objects that can be evaluated given the required variables. However, the equations themselves do not tell how the variables are arranged in the vectors. To use the equations correctly, entries in the state vector, the input vector, and the derivative of the input vector must be arranged correctly. Moreover, to be able to use the output vectors, correct interpretation is necessary. To interpret an output vector, how each `TwoTerminalDevice` (`SwitchableElements` in particular) was modeled must be known. Knowing the model corresponding to each device is not enough; we must also know which entries correspond to which devices. Additional information about the model is also stored in the `StateSpaceModel` object to help implement details of the model analysis processes.

State Space Model Manager Class

The `StateSpaceModelManager` is primarily a storage center for `StateSpaceModel`s. It is used by the `StateSpaceModelAnalyzer` to store models after their initial build. The `StateSpaceModelAnalyzer` interrogate this object before reconstructing a new `StateSpaceModel`, and uses the available models if possible. This object stores `StateSpaceModel`s indexed by the corresponding switch configuration representing the topology.

Switched Network Simulator Extension

The Switched Network Simulation Extension consists of the `DiscontinuityAnalyzer`, and extends the set of component models supported by the RLCM-NS. It is primarily responsible for managing the additional component models.

The extension supports `SwitchableElementController`s, which manages switch transitions, `DiscontinuityIndicator`s, which attach to independent sources, and `TimeLimit`s, which allow for more precise continuous time integration algorithms. This extension interacts with certain component models from the RLCM domain to manage piecewise linear time invariant intervals.

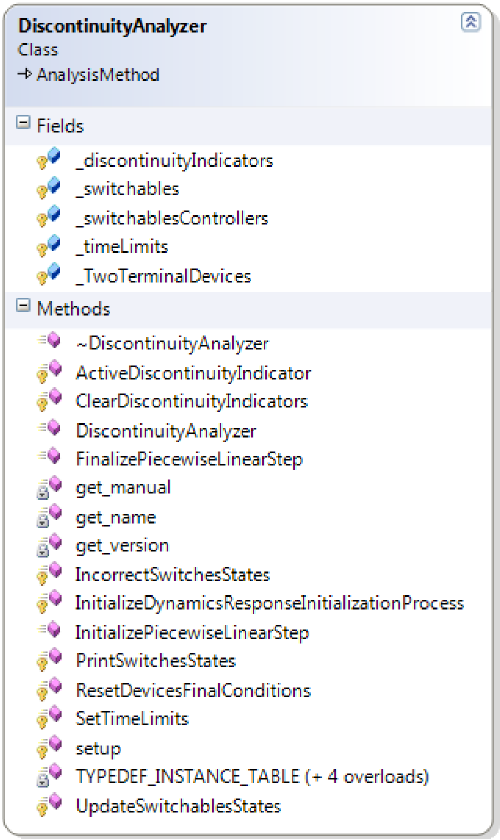
The `DiscontinuityAnalyzer` serves two main purposes, initializing and finalizing piecewise linear simulation steps.

The piecewise linear simulation step initialization process coordinates with the RLCM-NS in order to step from t_o^- to t_o^+ safely. This process simplifies the simulation loop invariant, and ensures that the numerical integration algorithm will give enough data to express the transition events precisely. The initialization process assumes that consistent final conditions are available. The initialization process finds the consistent switch states, handling all switch transitions, and then instructs the RLCM network simulator to take a zero length simulation step. Consistent switch states are found such that the switch control, as indicated by the SCS, is satisfied. The initialization process, thus, ensures that the RLCM-NS has all it needs to correctly simulate the next piecewise linear time interval.

The piecewise linear simulation step finalization process is a simple method to store the simulated data during the piecewise linear time interval such that they are accessible by the initialization process as consistent final conditions.

This extension converts switch transitions to discontinuities that the RLCM-NS can handle. This means that the numerical integration algorithm used by the simulator must support impulses, and for precision, it must support zero length integration steps. In this project, impulses are modeled exactly by abstracting the impulse value and operating on the impulse factors only. Thus, support for impulses in the integration algorithm is a trivial addition operation with proper object oriented design. Moreover, zero length steps are implemented by considering the impulses only in the integration.

Figure 4.2, and Figure 4.3 gives an overview to the most important classes in this extension.



DiscontinuityAnalyzer
Class
→ AnalysisMethod

⊟ Fields

- _discontinuityIndicators
- _switchables
- _switchablesControllers
- _timeLimits
- _TwoTerminalDevices

⊟ Methods

- ~DiscontinuityAnalyzer
- ActiveDiscontinuityIndicator
- ClearDiscontinuityIndicators
- DiscontinuityAnalyzer
- FinalizePiecewiseLinearStep
- get_manual
- get_name
- get_version
- IncorrectSwitchesStates
- InitializeDynamicsResponseInitializationProcess
- InitializePiecewiseLinearStep
- PrintSwitchesStates
- ResetDevicesFinalConditions
- SetTimeLimits
- setup
- TYPEDEF_INSTANCE_TABLE (+ 4 overloads)
- UpdateSwitchablesStates

Figure 4.2 Class diagram for Discontinuity Analyzer

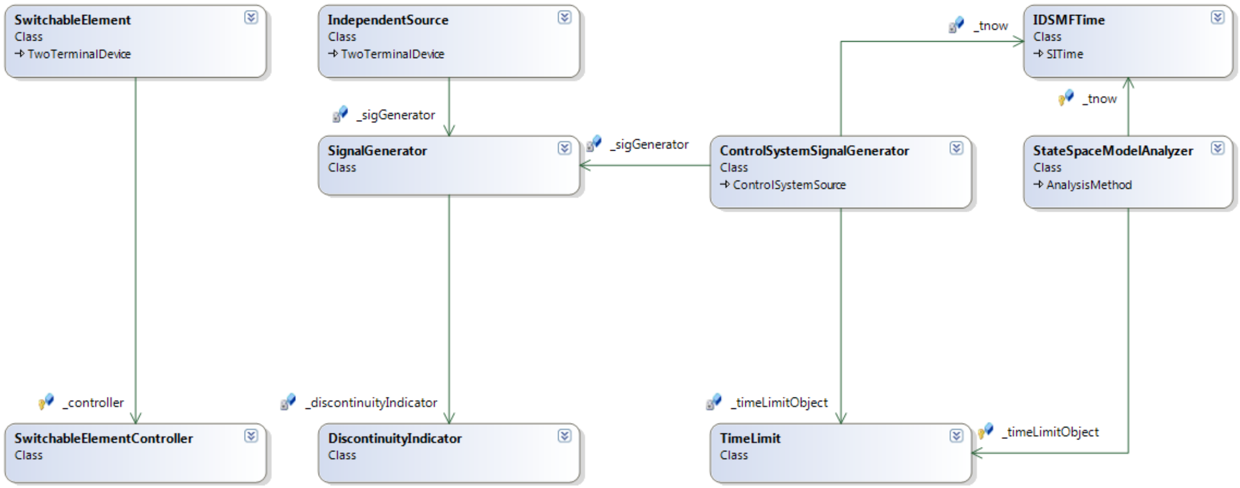


Figure 4.3 Class diagram for classes the Discontinuity Analyzer interacts with

Analysis Method: Discontinuity Analyzer Class

The `DiscontinuityAnalyzer` manages the analysis needed to turn the RLCM-NS into a switched network simulator. Its primary concern is switches. Switch transitions are modeled as discontinuities, which the `StateSpaceModelAnalyzer` sees as independent source discontinuities. Switch transitions are also affected by real discontinuities occurring in actual independent sources. The `DiscontinuityAnalyzer` therefore ensures that discontinuities of independent sources, and switch transitions are analyzed correctly. Because the `DiscontinuityAnalyzer` operates in the context of switched network simulation, it was assigned controlling `TimeLimit`s, and announcing discontinuities to allow dynamic `TwoTerminalDevices` to reset their final conditions.

The most critical role of the `DiscontinuityAnalyzer` is ensuring that switches are in consistent states before handing the control to the next analysis method. An iterative search algorithm is employed to find the consistent switch states. Because other analysis methods were designed to support impulses, and to support the dynamics of switched networks, the `DiscontinuityAnalyzer` operates as a commander in the search algorithm. While time and final conditions are frozen, this analysis method controls the `TimeLimit`, allowing for precise zero length simulation steps, and commands the `StateSpaceModelAnalyzer` to refresh its `StateSpaceModel`, and the `ContinuousTImeDynamics` to perform the zero length simulation steps. The `DiscontinuityAnalyzer` monitors impulses, and uses the unified `SwitchableElementController` interface to identify the search space and to identify reaching the (or a) consistent switch configuration.

Switch Control System Module

The Switch Control System adds the control system component models. The control component models can be decomposed into three sets. The first set includes elements responsible for feeding the control system with inputs. The second set includes control blocks distinguished by their respective input-output characteristics. The last set includes elements that form the interface between the control system and the switches to be controlled.

The SCS is primarily responsible for evaluating the control system outputs and making these outputs available for classes controlling or monitoring the switches. Unlike the other modules, the SCS does not have a central analysis method. Instead, each control component model is assigned the responsibility of executing its own characteristic equation, guided by the ISMF.

It is important to note that some elements at the boundary of the SCS have a vague classification. For example, the `SwitchableElementController`s can be considered a part of SCS or the SNSE. Such elements are referenced in any context as necessary.

4.3.3 Component Models

Electrical Network Components

Electrical network components map the physical electrical devices to software entities. At the heart of this set is the `TwoTerminalDevice` hierarchy. Other electrical network components are implemented to facilitate describing the 'net list' as part of the user interface.

`TwoTerminalDevice` hierarchy of devices plays the role of a user interface to specify the 'net list' of the network to be simulated. It also plays a role in abstracting unnecessary details. Moreover, the analysis methods consider them to model the network and then to interpret the simulation results.

Figure 4.4 gives an overview of the electrical network components' classes.

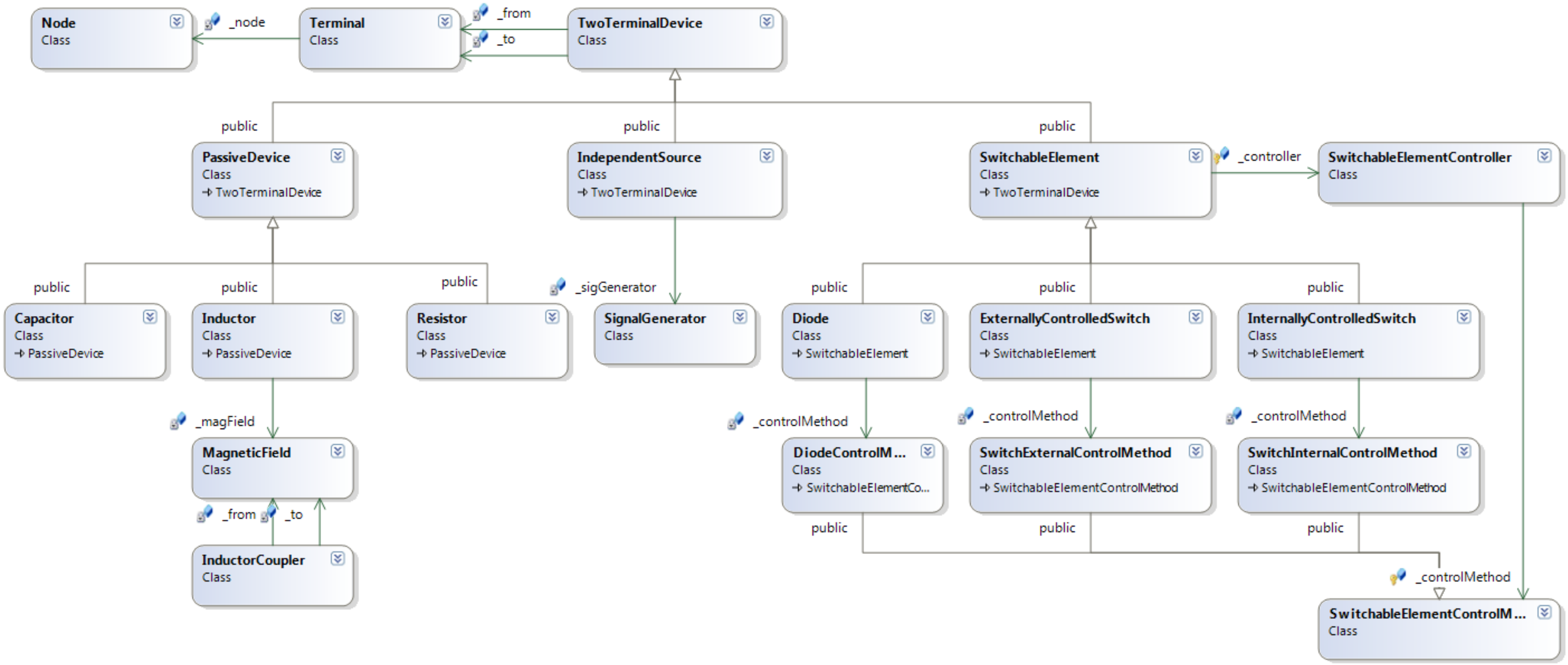


Figure 4.4 Class diagram for electrical network component models

Node Class

The `Node` class is used by the ISMF to connect `Terminal`s as specified by user input. In a physical switched RLCM network, a `Node` represents a node where wires are connected together. This class is very basic. Other than its use by the ISMF, it allows `TwoTerminalDevice`s to identify adjacent `TwoTerminalDevice`s, which is critical for analysis.

`TwoTerminalDevices` access `Node`s they are connected to through their `Terminal`s. Having access to the `Node`s they are connected to, they can identify adjacent `TwoTerminalDevice`s through the list maintained by the `Node`.

Terminal Class

`Terminal` represents a wire in a physical switched RLCM network. In the object oriented design context, they serve the purpose of connecting `TwoTerminalDevices` to `Node`s

Two Terminal Device Class and Hierarchy

The `TwoTerminalDevice` class serves as a base class for representing most of the devices in a switched RLCM network. The `TwoTerminalDevice` class includes two attributes of type `Terminal`, which are important when interconnecting devices to represent networks. The `TwoTerminalDevice` class also includes a voltage variable and a current variable. To allow the control system to access those two attributes, the class also includes an attribute of type `ControlSystemSourceTerminal`.

Classes from the hierarchy derived from `TwoTerminalDevice` are generally referenced through `TwoTerminalDevice` pointers. Derived class objects are referenced directly only when necessary; an enumerated type `DeviceType` is used to support this.

This object defines the virtual `ResetFinalCondition` used by the SNSE to finalize a piecewise linear simulation step. This function is defined by each child based on how it needs to reset its final conditions given the simulation data.

Passive Device Class and Hierarchy

The `PassiveDevice` class represents passive elements in a switched RLCM network; `Resistor`s, `Capacitor`s, and `Inductor`s.

Inductor Class

The `Inductor` class is more complex than the `Resistor` and `Capacitor` classes. An `Inductor` is defined by its inductance and the magnetic field it induces or it is affected by, which is how the mutual inductance and transformers are simulated.

Magnetic Field Class

`MagneticField`s are owned by `Inductor` class, and they keep a list summarizing the mutual couplings between `Inductor`s.

Inductor Coupler Class

`InductorCoupler` class is used primarily to allow user to define the mutual coupling relationships between Inductors. In principle, this object dictates how magnetic flux induces by an inductor affects another inductor in the network. Each `InducotrCoupler` is defined by two `Inductor`s, and a coupling coefficient.

Independent Source Class and Hierarchy

`IndependentSource`s have `SignalGenerator` objects. The generated signal by the `SignalGenerator` defines the characteristic of an `IndependentSource`.

Signal Generator Class

`SignalGenerator` component model is designed to generate mathematically defined signals. It is used as a part of other component models such as `IndependentSoures`. This class is highly configurable to meet user needs.

The `SignalGenerator` employs a verity of techniques to implement the set of functions it is capable of generating. It configures objects from the `SignalFunction` hierarchy, which implement mathematical signals, and make them available for consumer objects. For certain signals, such as the square wave, `ScheduledEvent` component model plays a critical role to allow for the precise discontinuities the wave requires. Therefore, at discontinuity instants, the `ScheduledEvent` component model helps generating the discontinuity. Because these discontinuities can affect switch configuration, the `SignalGenerator` object has `DiscontinuityIndicator` to announce when it has one.

Switchable Element Class and Hierarchy

`SwitchableElement` is a general representation for a switch. As the name suggests, it has a switchable Boolean state. It has a `SignalFunction` type attribute to allow abstracting switch transitions as discontinuities. In order to fully abstract switch transitions from the RLCM-NS, final conditions are handled in this object, and are transferred properly to the `SignalFunction` at transition instants before the `SignalFunction` is used as input in the state space model.

To allow for a variety of switchable elements, while hiding the unnecessary complexities, `SwitchableElementController` classes are used.

Switchable Element Controller Class

The `SwitchableElementController` hides the complex switch control complexities from `SwitchableElement`. It allows all component models down the `SwitchableElement` hierarchy to be analyzed in a uniform manner regardless of how they are being controlled. This class defines a common ground for The RLCM-NS, the SNSE, and the SCS. The `SwitchableElementController` has exclusive access to the state of the switch it controls, and it is assigned the responsibility to alter the switch state when commanded by the `DiscontinuityAnalyzer`.

The RLCM-NS relies on the `DiscreteEvent` objects managed by this component model to identify time intervals free of switch transitions. The SNSE uses the utility provided by this object to realize switch transitions, and when it attempts to find consistent switch states. More importantly, the SCS affects the simulation process through `SwitchableElementControlMethod` this object has and uses to update the state of `SwitchableElement`s.

Switchable Element Control Method Class and Hierarchy

This object defines the interface `SwitchableElementController` uses to find the state the `SwitchableElement` must be in. The specific method is defined by the children of this object.

Externally Controlled Switch Class

An `ExternallyControlledSwitch` is switch that can be controlled from the '.scn' file only. To implement this control method, the `SwitchableExternalControlMethod` is used.

Switchable External Control Method Class

This class is very simple. It has a controller state variable which can be altered only by the ISMF as dictated by the '.scn' file.

Internally Controlled Switch Class

The state of this object is dependent on the switch control system outputs. To implement this control dependency, the `SwitchInternalControlMethod` is used.

Switch Internal Control Method Class

This control method interfaces switches with the SCS. The control state of switches that uses this object is obtained from the SCS outputs through `ControlSystemDigitalSinkTerminal`.

Diode Class

`Diode` can be represented as an `InternallyControlledSwitch`. However, because of its popularity, and because it is a common element, it was implemented as an independent component model. To control the state of the `Diode`, the `DiodeControlMethod` is used.

Diode Control Method Class

The `DiodeControlMethod` has direct access to the `Diode` voltage, current, and present state. It uses the `Diode` variables to determine the state it must be in.

Control System Components

The control system component models are shown in Figure 4.5, and Figure 4.6 shows the dependencies between control system and electrical network component models.

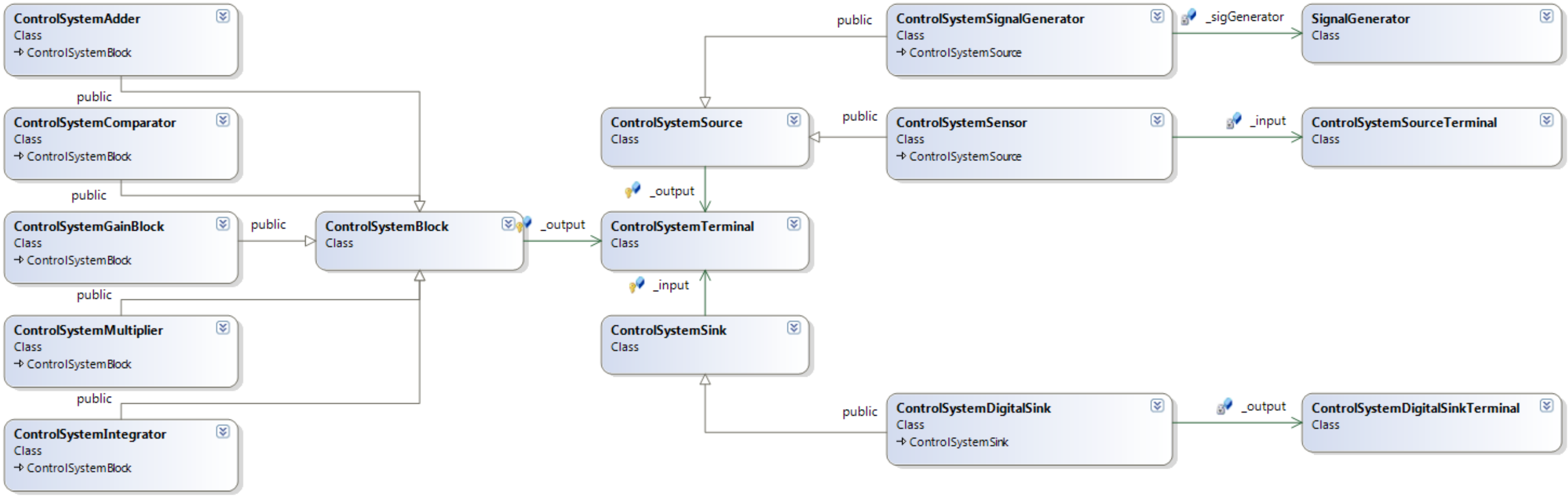


Figure 4.5 Class diagram for control system component models

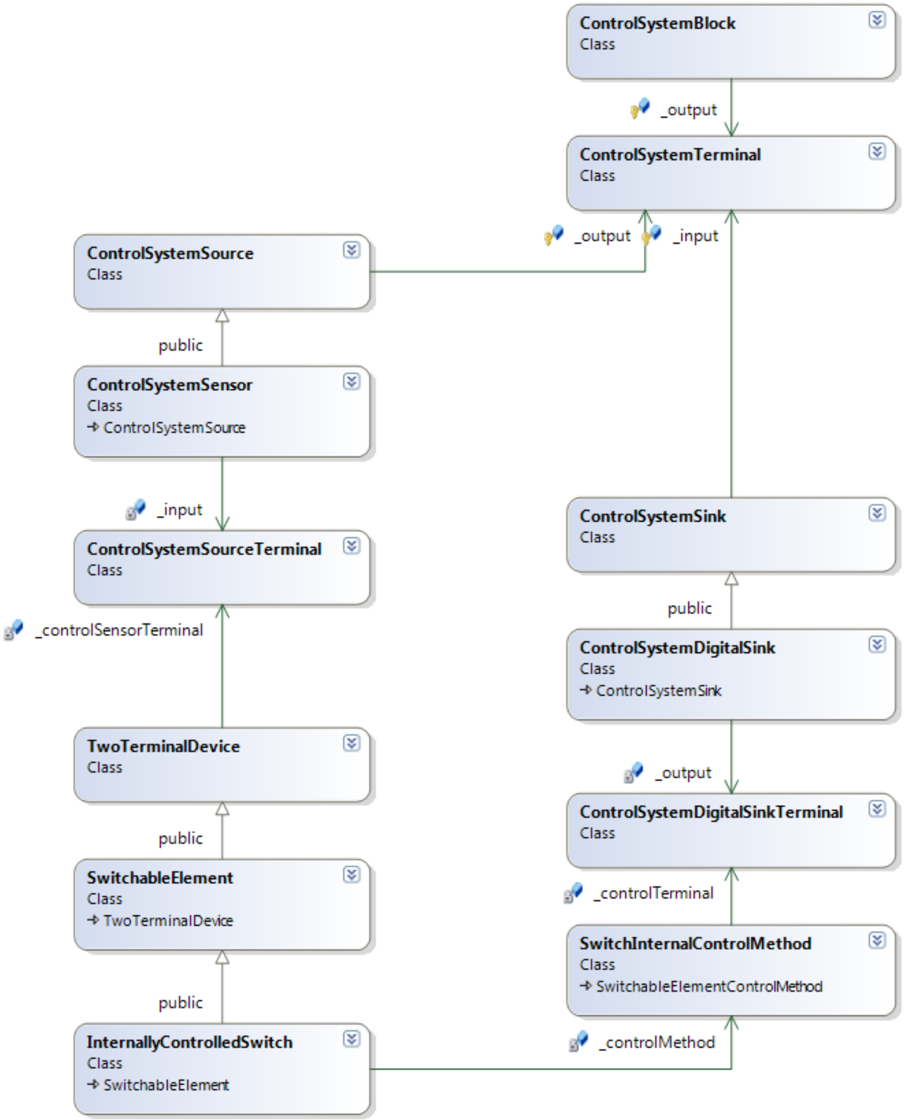


Figure 4.6 Class diagram showing the interface between control system and electrical network component models

Control System Block Class and Hierarchy

The `ControlSystemBlock` represent a mathematical control operation specified by an input-output characteristic. The child component models are used to represent the body of the control system. Each block owns its output terminal of `ControlSystemTerminal` type. The output terminal is used to transmit the result, or the output signal, such that it is accessible by consumer blocks. Inputs to each block are defined by each child block since the number of inputs is variable.

The other important aspect of this component model is an `InstanceFunction` it has, which encapsulates the characteristic function of the block. Each block submits its function to the ISMF specifying the input and output dependencies. The ISMF then helps the SCS find the control outputs.

Control System Source Class and Hierarchy

Control System Source Class

The `ControlSystemSource` represents the input interface to the SCS. Similar to the `ControlSystemBlock`, the `ControlSystemSource` has an output `ControlSystemTerminals` and an `InstanceFunction`. The `ControlSystemTerminals` serves an identical purpose to that used in `ControlSystemBlock`, and the `InstanceFunction` is defined by the sourcing method.

Control System Sensor Class

The `ControlSystemSensor` sourcing input to the SCS from the power stage. It supports reading voltage or current, which is accessed from `TwoTerminalDevices` through the `ControlSystemSourceTerminal` attached to them.

Control System Signal Generator Class

`ControlSystemSignalGenerator` sources input from a signal generated using predefined functions, which can be configured by the user through the '.ism' file. Similar to `IndependentSources`, this component model uses a `SignalGenerator` object to manage signal generation processes.

Control System Sink Class and Hierarchy

The `ControlSystemSink` is the output interface of the SCS. For the purposes of this project, this interface is limited to controlling switches. Each `ControlSystemSink` has an input `ControlSystemTerminals`, which has access to the control signal to be transmitted at the output terminal. The `ControlSystemSink` has an `InstanceFunction`, similar to all other SCS component models. The function is used to transfer the input control signal to the desired format at the output terminal.

Control System Digital Sink Class

The `ControlSystemDigitalSink` converts the input signal to a digital signal that can be used to control switches. `SwitchableElementControllers` access the digital control signal through the output `ControlSystemDigitalSinkTerminal of the ControlSystemDigitalSink.`

Control System Terminal Class

The `ControlSystemTerminal` allows connecting `ControlSystemBlocks`, `ControlSystemSources`, and `ControlSystemSinks` to form the desired switch control system. They are used to transmit signals from the producer of the signal to the consumer.

Control System Source Terminal Class

The `ControlsystemSourceTerminal` is designed to attach to `TwoTerminalDevices` and to be accessible by `ControlSystemSensors` in order to allow voltages and currents to be transmitted to the SCS.

Control System Digital Sink Terminal Class

The `ControlSystemDigitalSinkTerminal` transmits digital control signals to be made available to `SwitchableElementControllers`.

4.3.4 Impulse support and the Signal Value object

`SignalValue` class was designed to implement the support for impulses in the analysis. This class defines the basic representation for impulses. It also defines the arithmetic needed by the analysis methods and the numerical integration algorithm. The numerical integration algorithm was updated to use the added arithmetic operations.

4.4 Objects Interaction (Sequence Diagrams)

This section presents the interactions and the processes between or within the important classes in the simulation process. The interactions and processes are presented as UML sequence diagrams.

4.4.1 Compute Dynamic Response

`ComputeDynamicResponse` is the main body of the simulation loop (as function of time). The ISMF calls it to make a simulation step of a finite time length. Figure 4.7 shows the sequence diagram for this method.

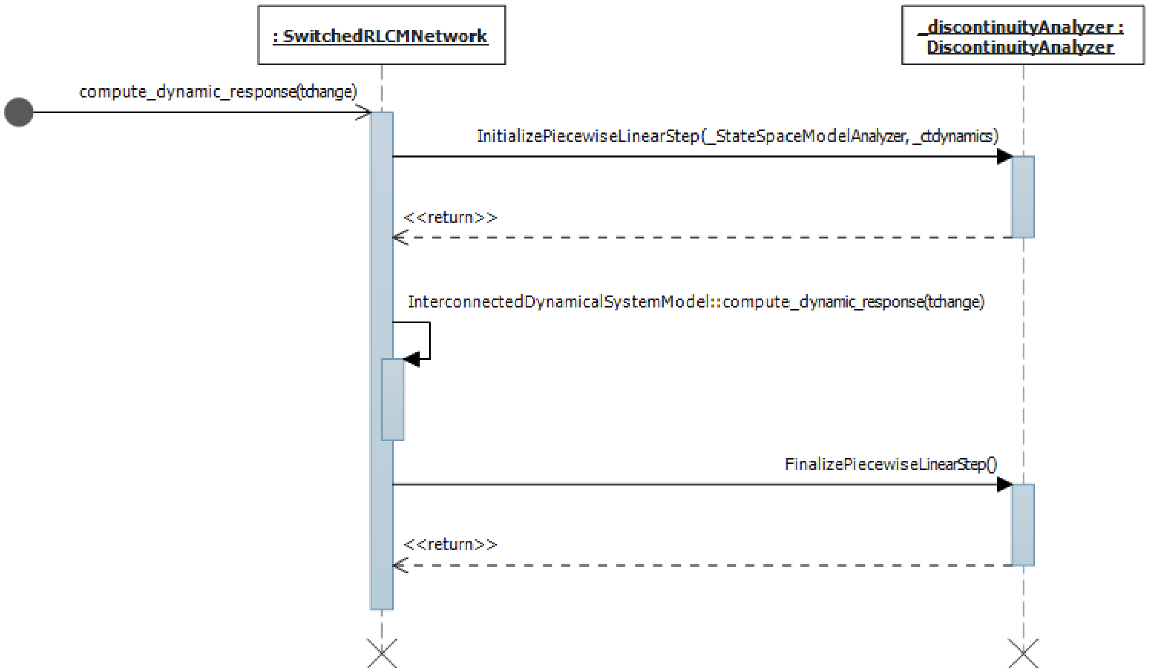

: SwitchedRLCMNetwork
_discontinuityAnalyzer : DiscontinuityAnalyzer
compute_dynamic_response(tchange)
InitializePiecewiseLinearStep(_StateSpaceModelAnalyzer, _ctdynamics)
<<return>>
InterconnectedDynamicalSystemModel::compute_dynamic_response(tchange)
FinalizePiecewiseLinearStep()
<<return>>

Figure 4.7 Sequence diagram for Compute Dynamic Response

4.4.2 Initialize and Finalize Piecewise Linear Step:

The `InitializePiecewiseLinearStep` and `FinalizePiecewiseLinearStep` are at the interface to the `DiscontinuityAnalyzer`. Their operation is critical to abstracting a switched RLCM network into an RLCM network. Between the initialization and finalization, the network is seen as an RLCM network. Figure 4.8 presents `InitializePiecewiseLinearStep`, and Figure 4.9 presents `FinalizePiecewiseLinearStep`.

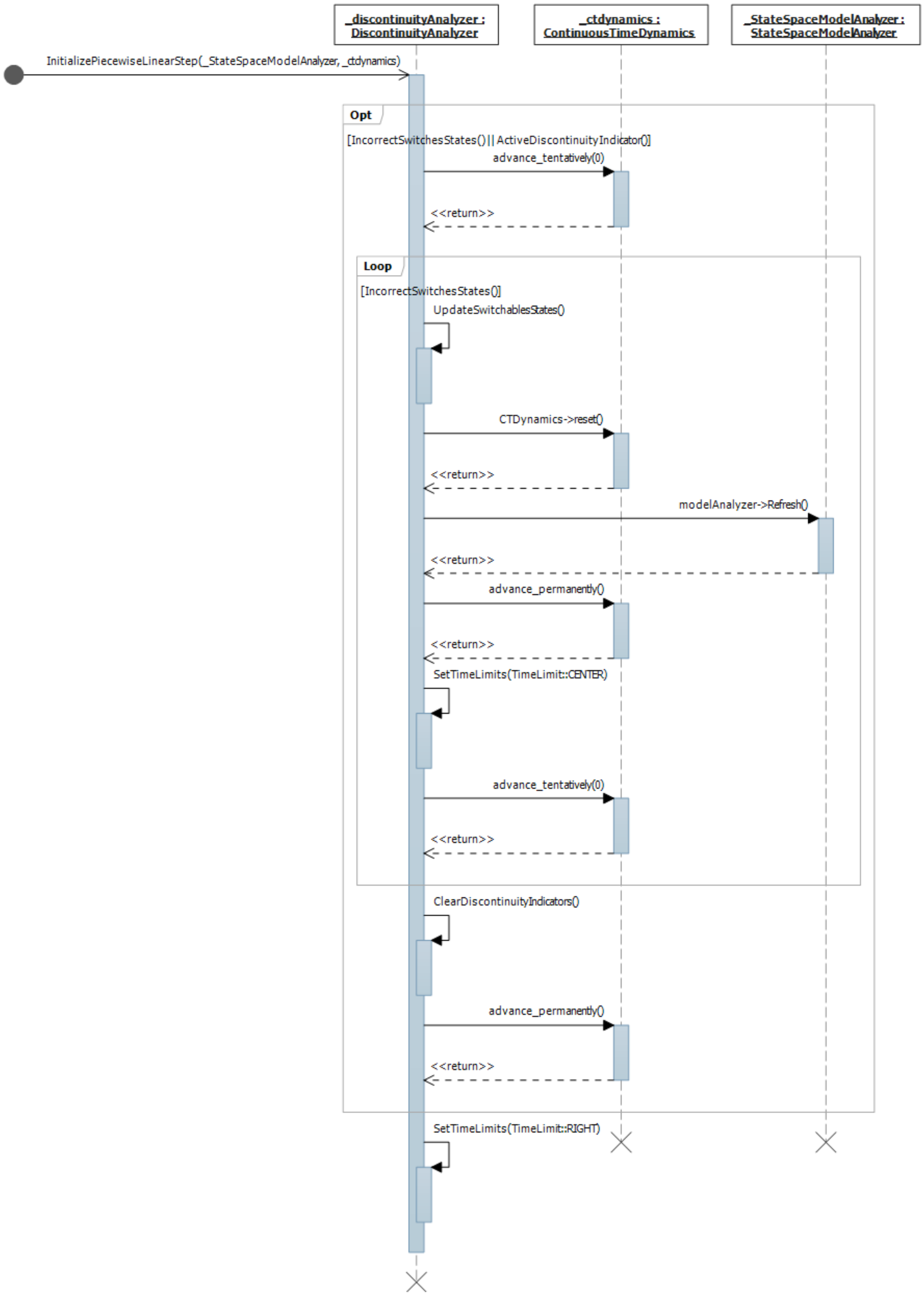

_discontinuityAnalyzer : DiscontinuityAnalyzer
_ctdynamics : ContinuousTimeDynamics
_StateSpaceModelAnalyzer : StateSpaceModelAnalyzer
InitializePiecewiseLinearStep(_StateSpaceModelAnalyzer, _ctdynamics)
Opt
[IncorrectSwitchesStates() || ActiveDiscontinuityIndicator()]
advance_tentatively(0)
<<return>>
Loop
[IncorrectSwitchesStates()]
UpdateSwitchablesStates()
CTDynamics->reset()
<<return>>
modelAnalyzer->Refresh()
<<return>>
advance_permanently()
<<return>>
SetTimeLimits(TimeLimit::CENTER)
advance_tentatively(0)
<<return>>
ClearDiscontinuityIndicators()
advance_permanently()
<<return>>
SetTimeLimits(TimeLimit::RIGHT)

Figure 4.8 Sequence diagram for Initialize Piecewise Linear Step

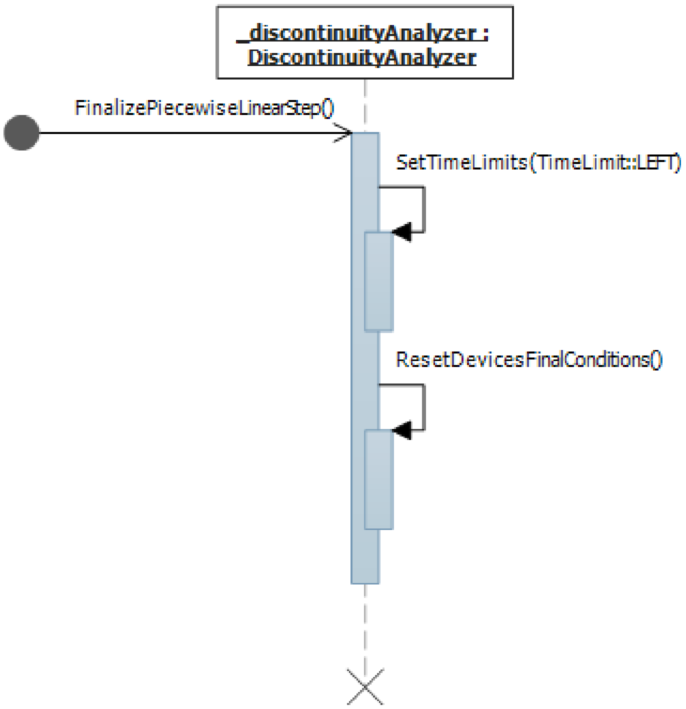

_discontinuityAnalyzer : DiscontinuityAnalyzer
FinalizePiecewiseLinearStep()
SetTimeLimits(TimeLimit::LEFT)
ResetDevicesFinalConditions()

Figure 4.9 Sequence diagram for Finalize Piecewise Linear Step

4.4.3 Refresh State Space Model

`Refresh` is a method in `StateSpaceModelAnalyzer`. It reflects the current configuration of the network into the state space model and makes it ready for use. Figure 4.6 shows the sequence diagram for this method.

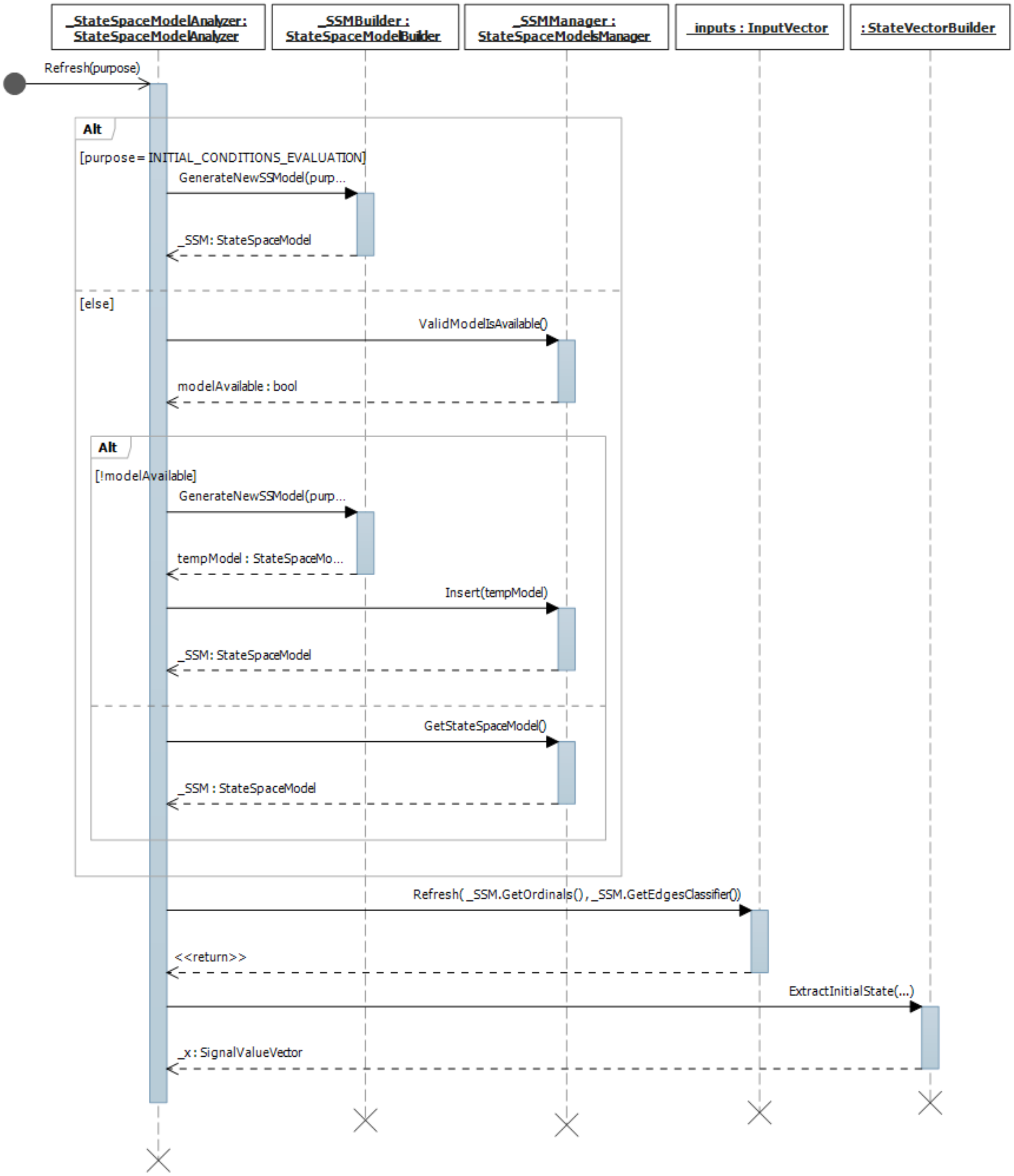

_StateSpaceModelAnalyzer : StateSpaceModelAnalyzer
_SSMBuilder : StateSpaceModelBuilder
_SSMManager : StateSpaceModelsManager
inputs : InputVector
: StateVectorBuilder
Refresh(purpose)
Alt
[purpose = INITIAL_CONDITIONS_EVALUATION]
GenerateNewSSModel(purp...
_SSM: StateSpaceModel
[else]
ValidModelIsAvailable()
modelAvailable : bool
Alt
[!modelAvailable]
GenerateNewSSModel(purp...
tempModel : StateSpaceMo...
Insert(tempModel)
_SSM: StateSpaceModel
GetStateSpaceModel()
_SSM : StateSpaceModel
Refresh(_SSM.GetOrdinals() , _SSM.GetEdgesClassifier())
<<return>>
ExtractInitialState(...)
_x : SignalValueVector

Figure 4.10 Sequence diagram for state space model Refresh

Chapter 5 Tests and Results

5.1 Simulation Tests and Results

In Chapter 4, the simulation software design was presented. This chapter presents test cases and results from the implemented simulator. The test cases are chosen to illustrate some of the simulator capabilities and to demonstrate correctness of the algorithms. The test cases are divided into three sets; simple switched networks, a rectifier network, and buck converter networks. Parameters, ism, and scn files are provided in Appendix A-B.

5.2 Simple Networks

Test Case 1 Switched Active RC Network

In this first test case, a simple externally controlled switch is used. It controls connectivity between a voltage source and a capacitor. When the switch turns ON, energy flows through the switch to charge the capacitor instantaneously. When the switch is OFF, the capacitor becomes part of an RC circuit and it discharges its energy. This network is used because it illustrates the basic functionality of ideal switches, the simulation of simple RC circuits, and a simple case of state discontinuity and re-initialization. Figure 1.1 shows the network, and Figure 5.1 shows the simulation results.

From Figure 5.1, it can be verified that the capacitor voltage experienced a step at the instant when the switch was closed, and was reinitialized to the proper voltage (equals source voltage). When the switch opened again, the capacitor voltage decays with time constant: $\tau = R\ C = \frac{1}{5}$.

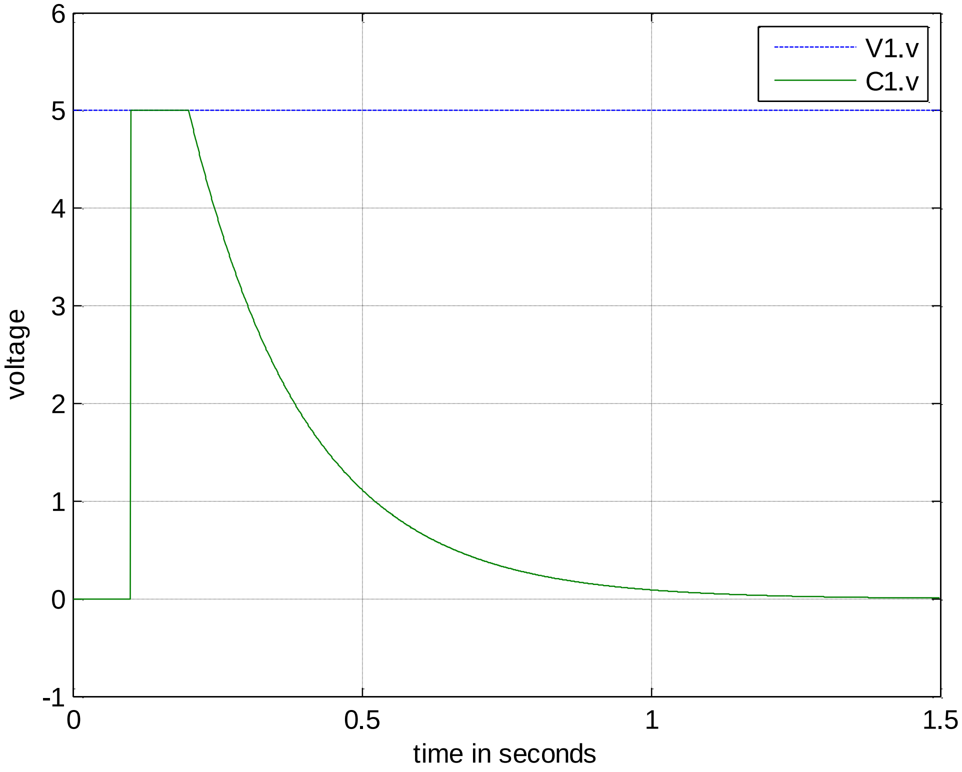


Figure 5.1 Simulation results for network in Figure 1.1

Test Case 2 Switched Capacitor Loop

This test case represents a very basic switched capacitor network. It illustrates a more interesting state discontinuity and re-initialization case. This case is also an example of initializing energy storing variables at the start of simulation. Figure 5.2 shows the network, and Table 5.1 shows the simulation data.

The results matches the expected theoretical values, where $v_{c1}(t_o^+) = v_{c2}(t_o^+)$ and $i_{c1}(t_o) = -i_{c2}(t_o)$ [or equivalently: $C_1\left(v_{c1}(t_o^+) - v_{c1}(t_o^-)\right) = -C_2\left(v_{c2}(t_o^+) - v_{c2}(t_o^-)\right)$].

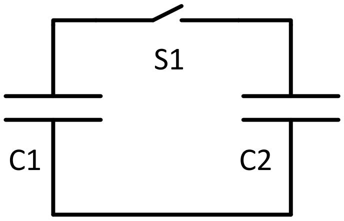


Figure 5.2 Switched capacitor loop

Table 5.1 Simulation results for network in Figure 5.2

Variable	Value at t_o^-	Value at t_o^+
S1 State	OFF	ON
C1.V	20	10
C2.V	5	10

5.3 Rectifier Network

Test Case 3 Bridge Rectifier Network

This test case simulates a full wave rectifier with a smoothing capacitor. It illustrates the operation of diodes and how the simulator manages their states automatically. Figure 5.3 shows the network. The results are plotted and compared to Generalized State Space Modeling simulation[6] in Figure 5.4, Figure 5.5, Figure 5.6, and Figure 5.7.

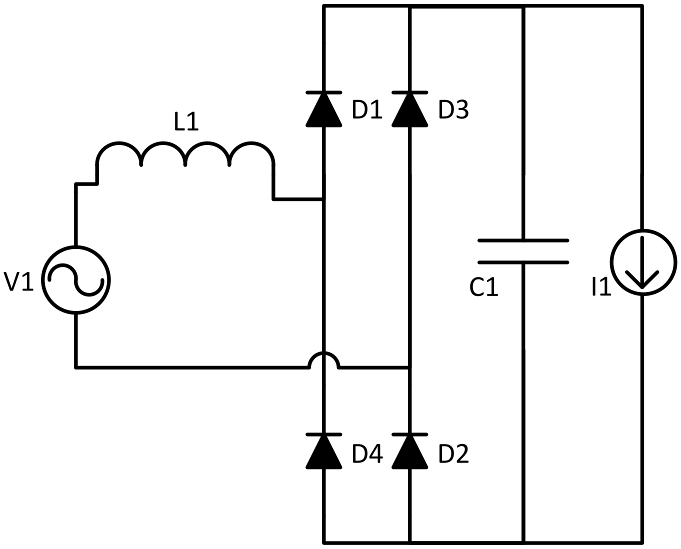


Figure 5.3 Bridge rectifier network

---

[6] Generalized State Space Modeling, and the term General State Space Model (used in Chapter 6) are different and not related.

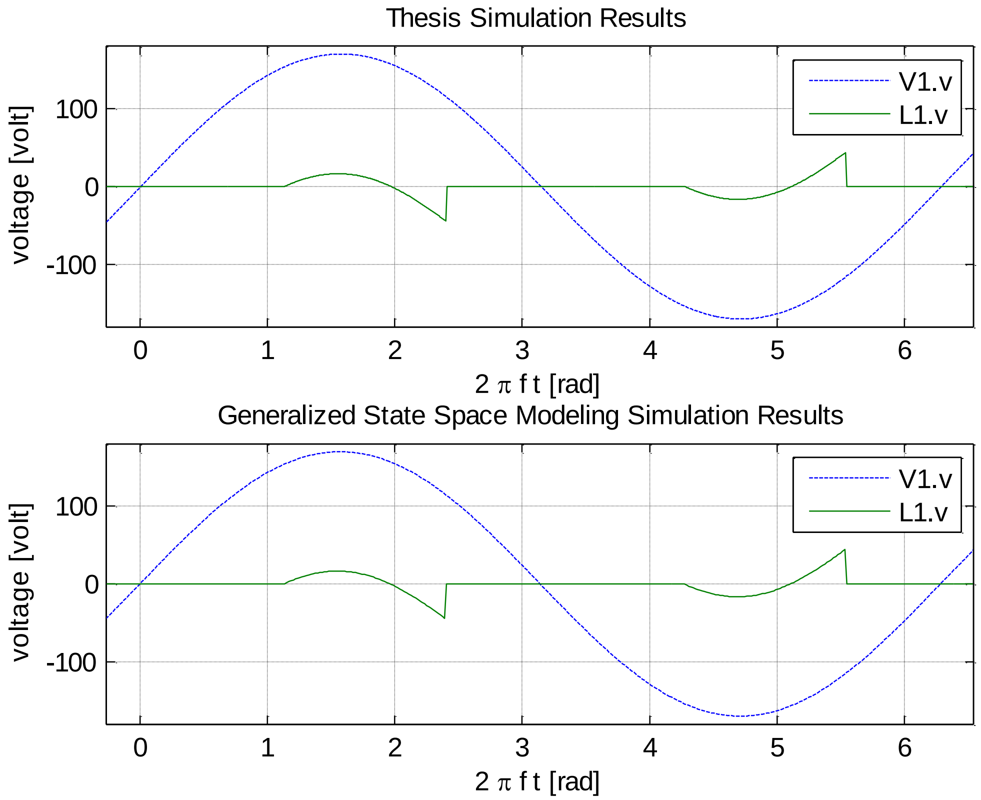


Figure 5.4 Simulated AC side voltage values for network in Figure 5.3

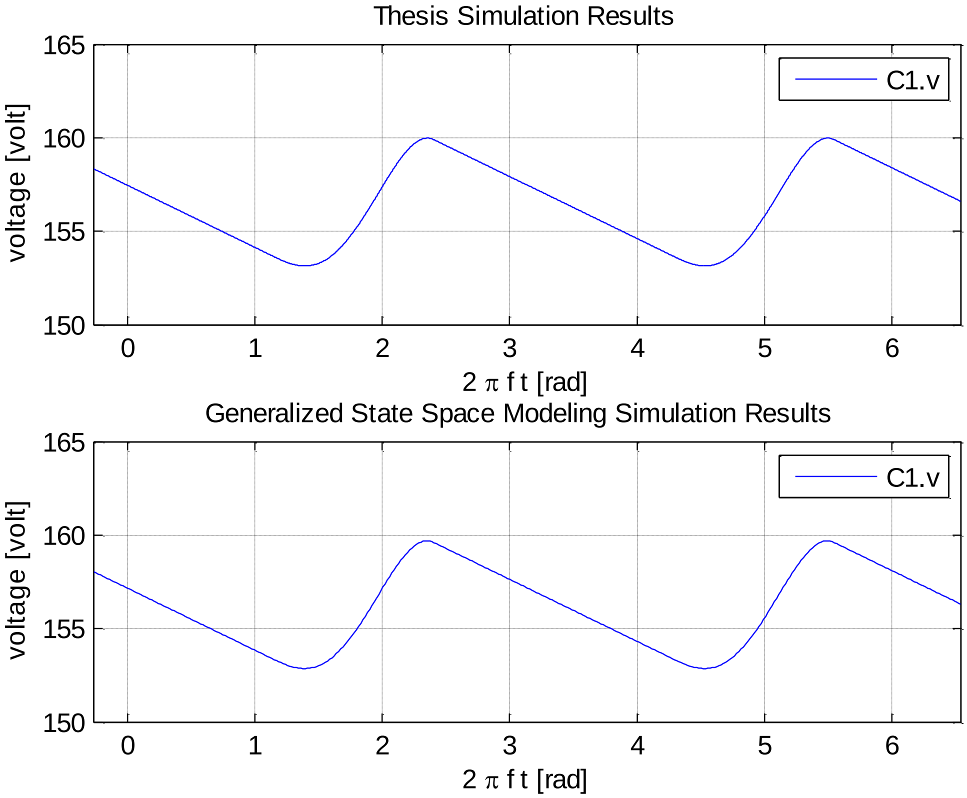


Figure 5.5 Simulated DC side voltage values for network in Figure 5.3

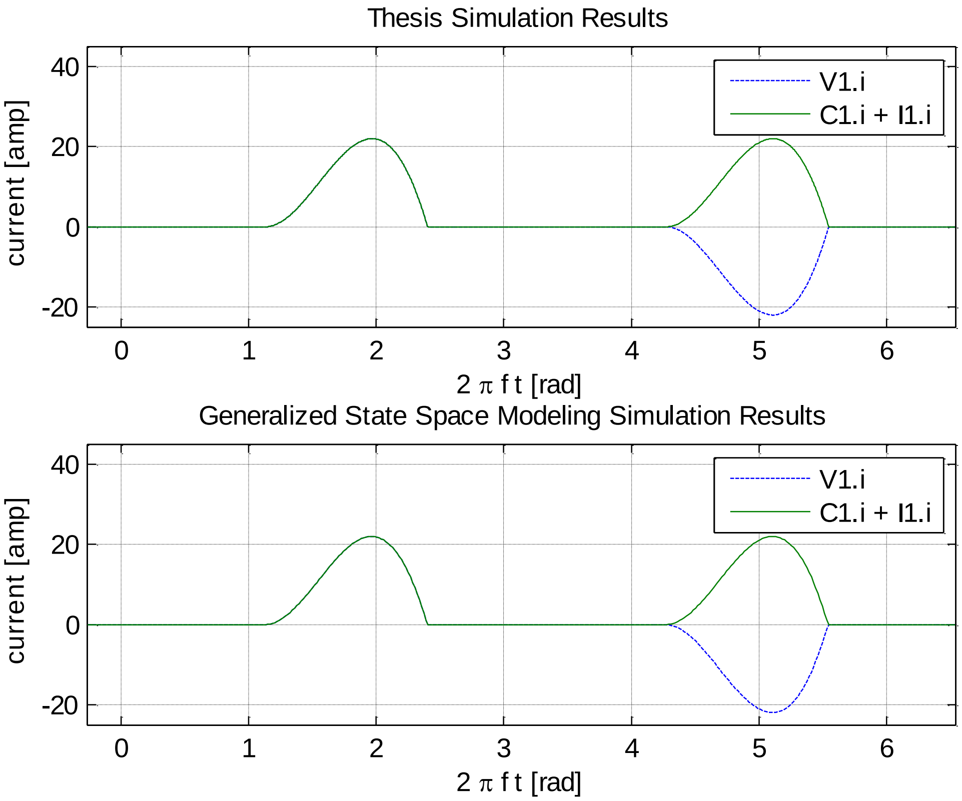


Figure 5.6 Simulated currents of the network in Figure 5.3

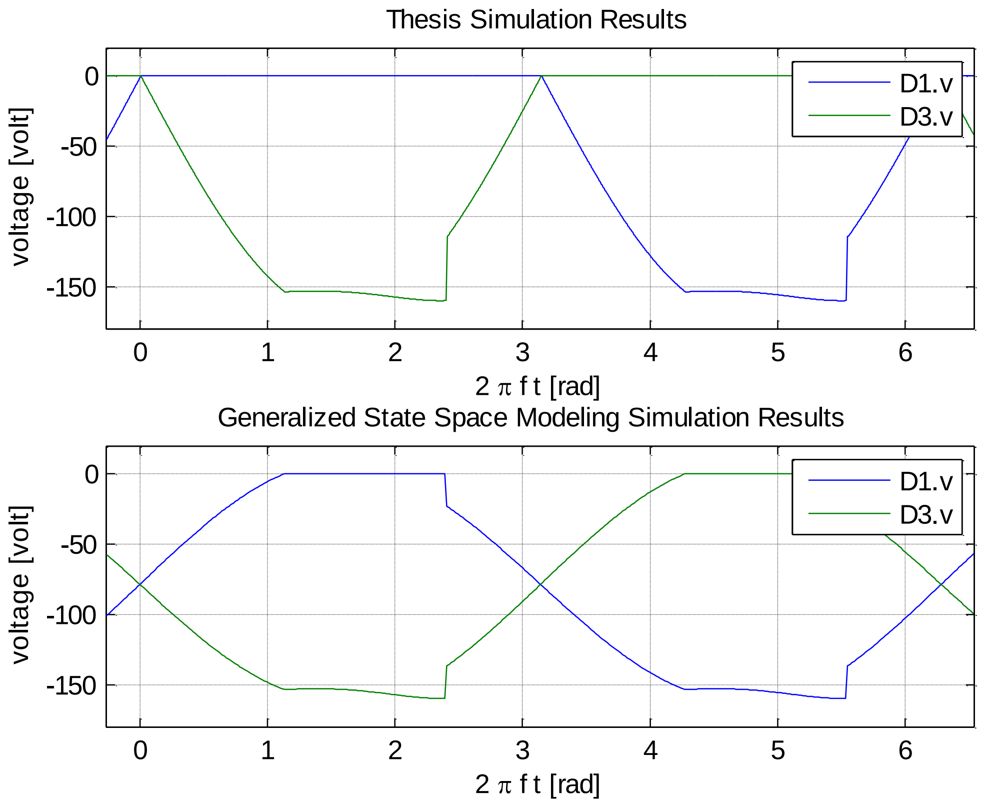


Figure 5.7 Simulated diode voltages of the network in Figure 5.3

Some differences are noticeable in Figure 5.7 when all diodes are in OFF state. This discrepancy is a result of the fact that in this mode, the dc-side is floating (does not have a common ground with the ac-side) as illustrated in Figure 5.8. Each simulator handles this case differently. However, they all must agree to the fact that the voltage drop across D1-D3 equals the voltage drop across V1-L1 as in Figure 5.9.

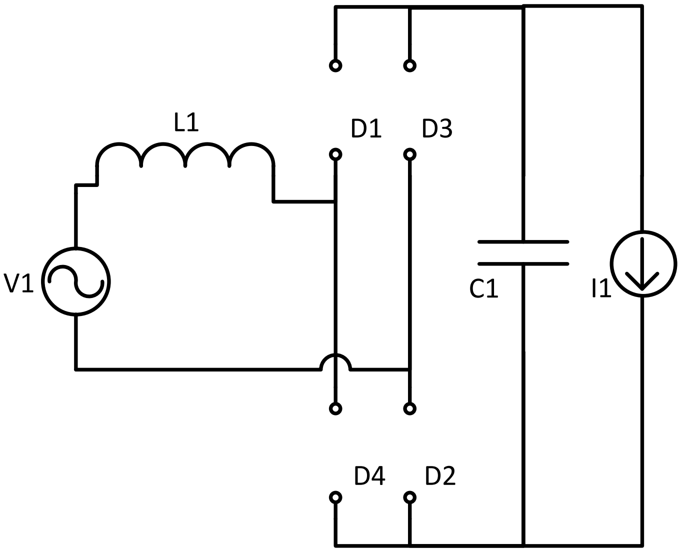


Figure 5.8 Bridge rectifier when all diodes are OFF

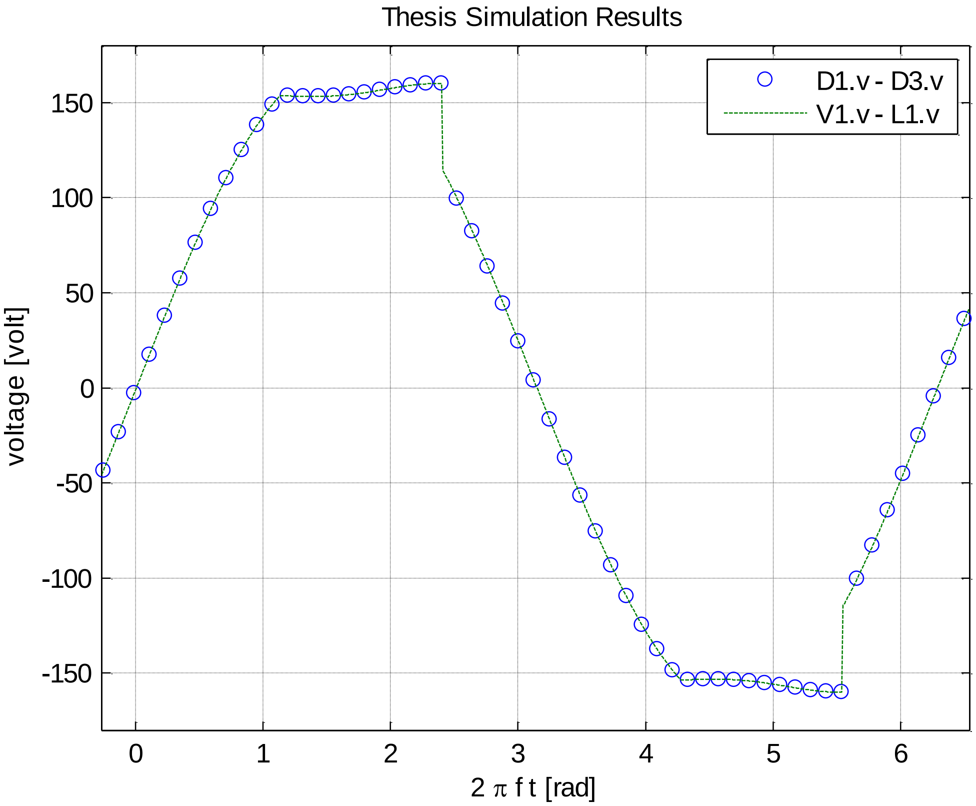


Figure 5.9 Consistent loop voltages in any switch configuration

5.4 Buck Converter Networks

Test Case 4 Buck Converter with constant switching frequency

This test case demonstrates the simulation results of a DC to DC buck converter. It illustrates the operation of internally controlled switches. The switch is controlled by a constant frequency clock representing a very simple control system. Figure 5.10 shows the network, Figure 5.11 shows the transient simulation results, and Figure 5.12 gives the outputs in the steady state.

In the steady state, the outputs match the theoretical results given in Table 5.2.

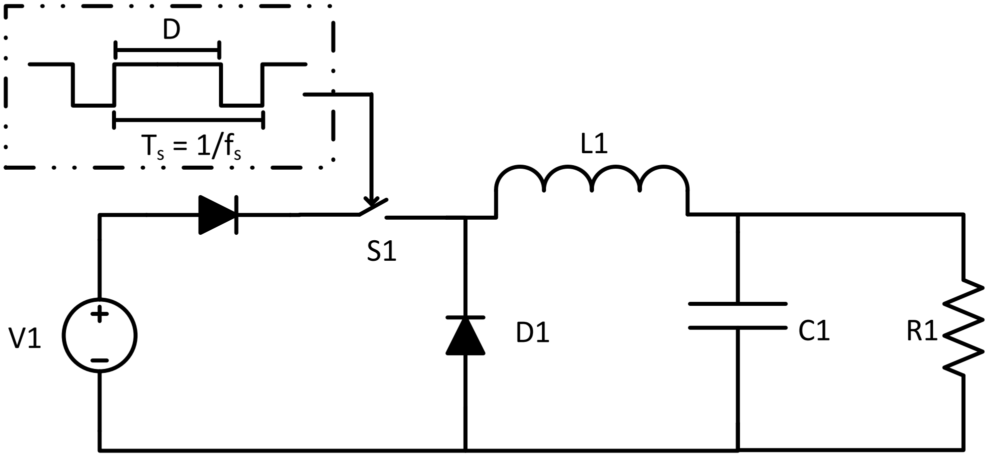


Figure 5.10 Buck converter with constant switching frequency

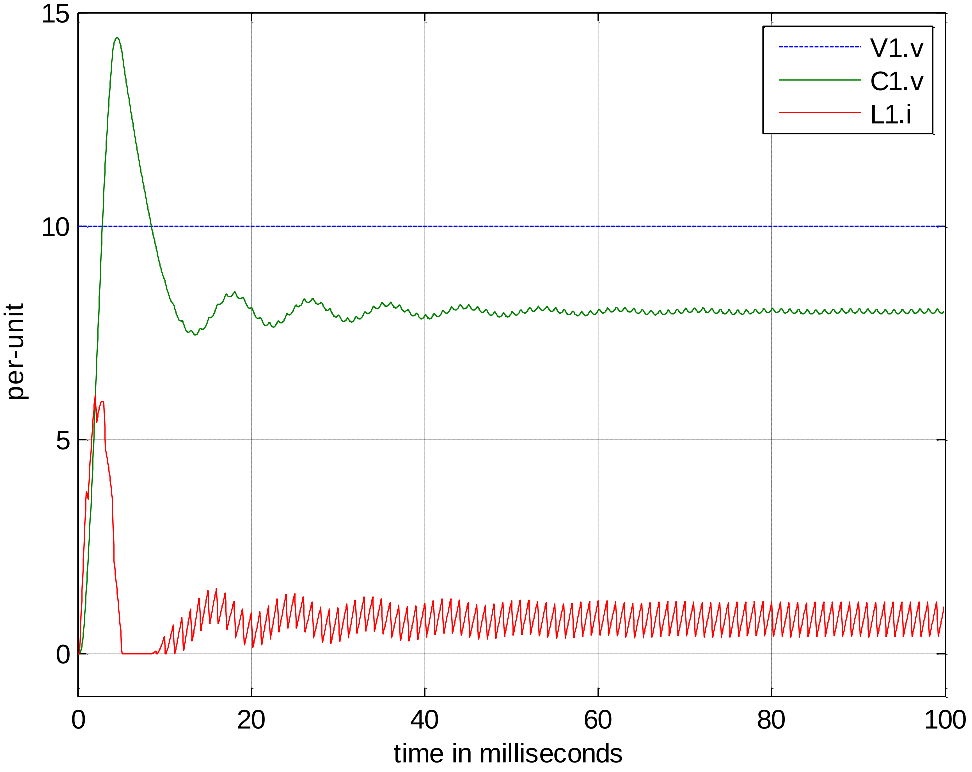


Figure 5.11 Simulation results for network in Figure 5.10

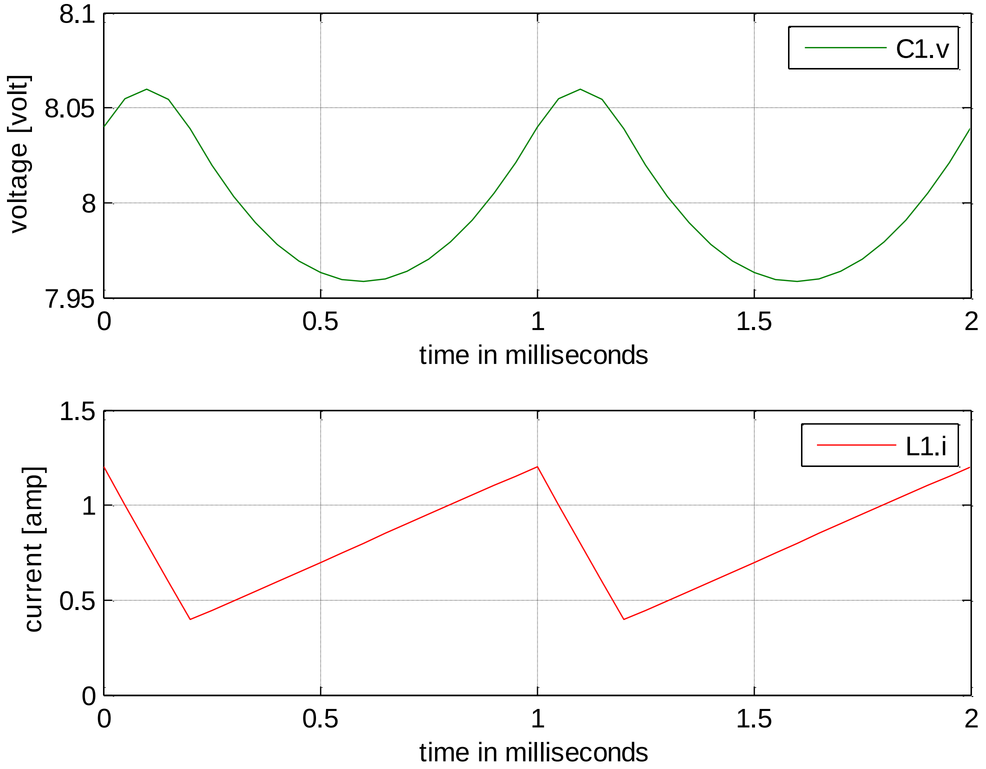


Figure 5.12 Simulation results for network in Figure 5.10 during steady state operation

Table 5.2 Theoretical results for network in Figure 5.10 during steady state operation

Variable	Equation	Value
$\boldsymbol{average(v_C)}$	DV_S	$8\,V$
$\boldsymbol{\Delta v_C}$	$DV_S\dfrac{1-D}{8\,L\,C\,f_s^2}$	$0.1V$
$\boldsymbol{\Delta i_L}$	$DV_S\dfrac{1-D}{L\,f_s}$	$0.8A$

Test Case 5 Buck Converter with feedback control system

This test case demonstrates the simulation results of a DC to DC buck converter. This buck converter uses a feedback control system. It demonstrates the functionality of a fairly complex network composed of power and control stages. The power stage is not very different from the earlier test cases. However, the control system this network implements uses various control system components from the available models. The control system uses a voltage sensor, adders, multipliers, an integrator, a comparator, a signal generator, and a switch controller (digital sink). Figure 5.13 shows the network, and Figure 5.14 through Figure 5.16 show the simulation results. The results are compared against Generalized State Space Modeling simulation results[7].

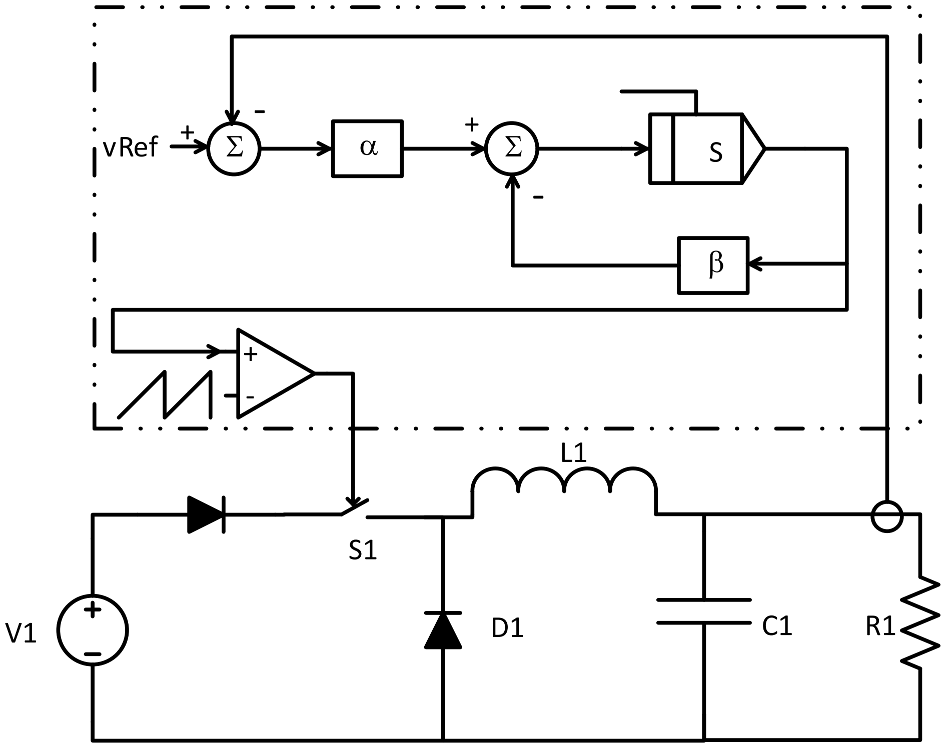


Figure 5.13 Buck converter with closed loop switch control

---

[7] Generalized State Space Modeling, and the term General State Space Model (used in Chapter 6) are different and not related.

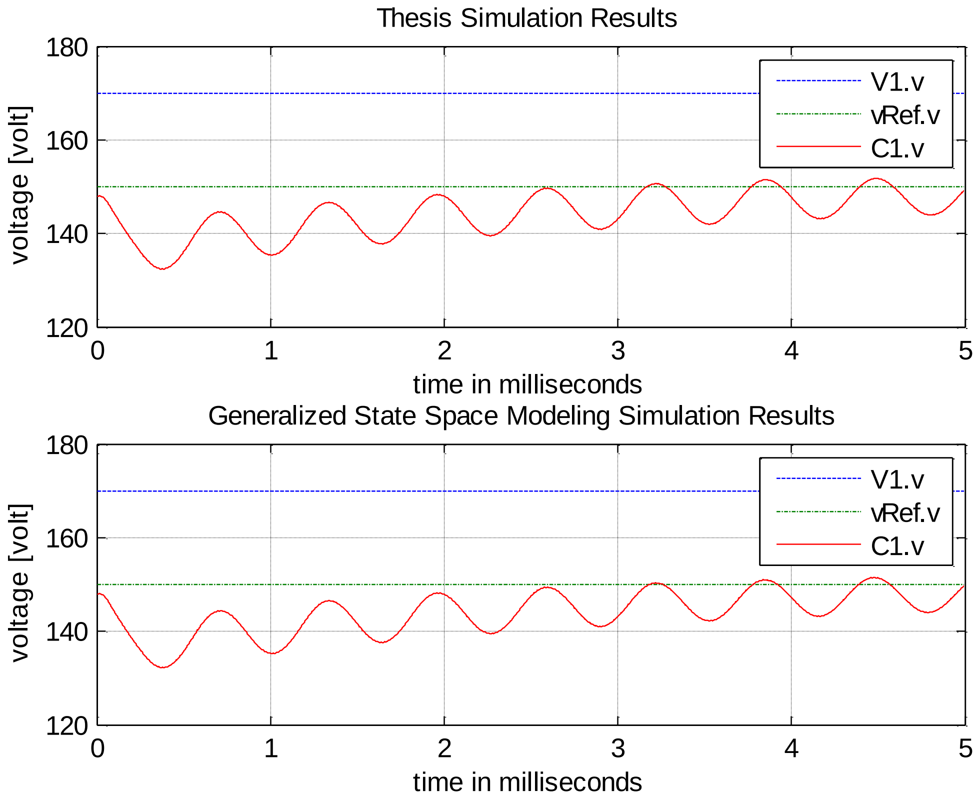


Figure 5.14 Input and output voltages for network in Figure 5.13

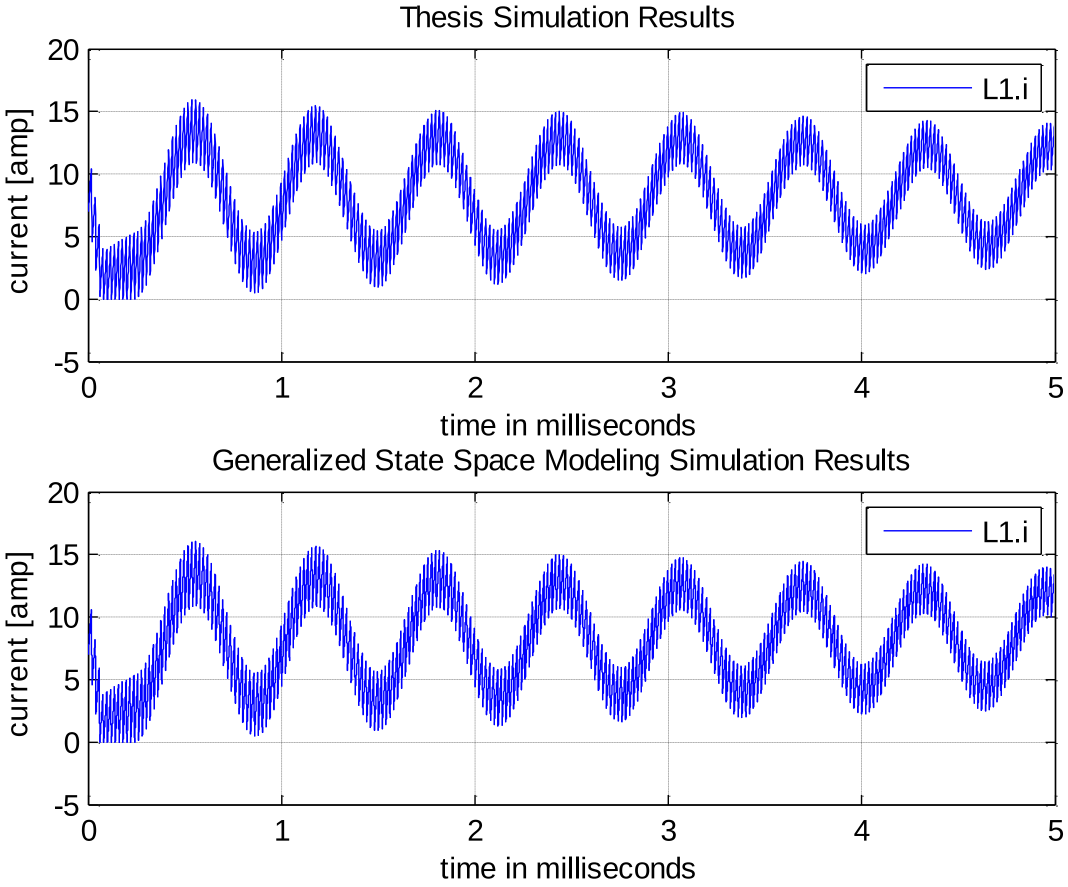


Figure 5.15 Output current for network in Figure 5.13

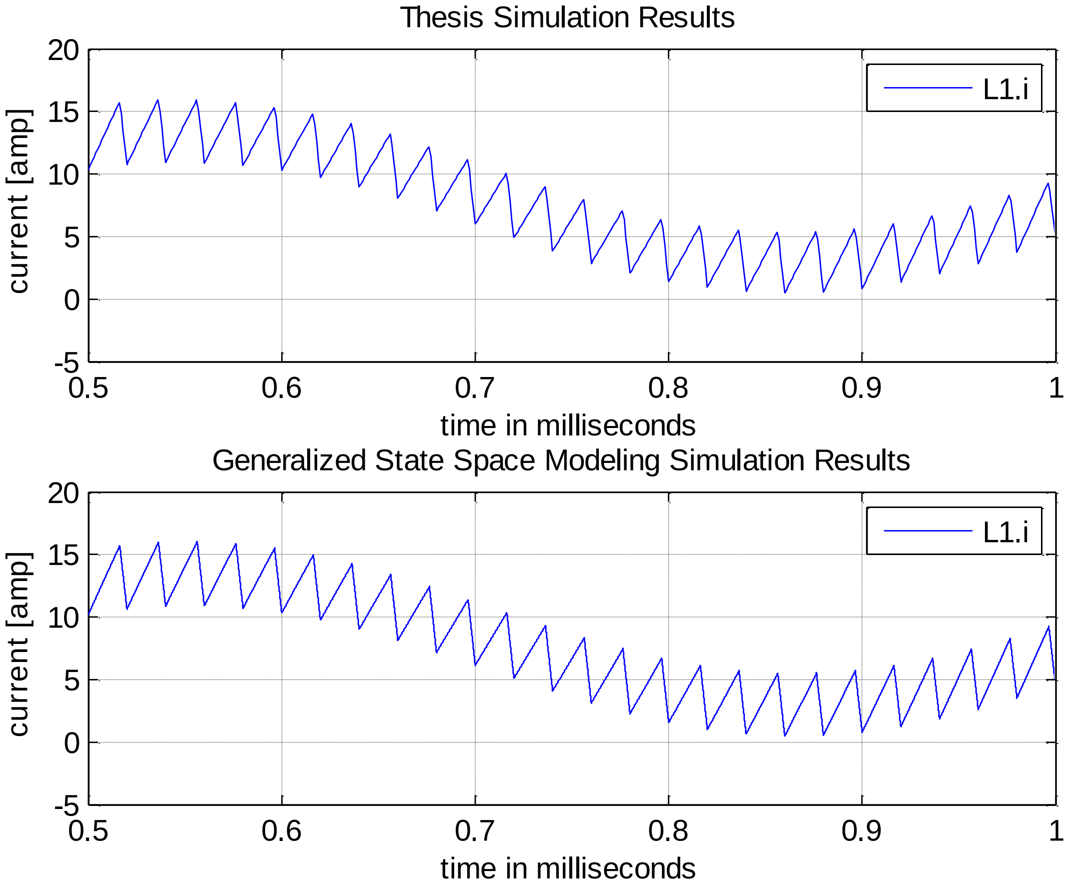


Figure 5.16 Output current for network in Figure 5.13 showing waveform details

Chapter 6 Future Work

6.1 Introduction

In this chapter, a promising future work based on the theory presented in the earlier chapters will be introduced. This thesis presented a more complete switch characteristic, which extended the validity interval of state space models eliminating the need for additional complex analysis. This chapter will attempt to introduce a possible extension as such to make a single state space model valid for all times. The method will be presented in a future work theme in order to introduce the interesting aspect of it.

6.2 Motivation

Switched RLCM Networks were analyzed and simulated traditionally using a sequence of state space models. Analyzing the sequence require generating a state space model for each topology. Moreover, because the state space models are valid in mutually exclusive time intervals, finding the correct topology after a switch transition could be an involved process.

Having a state space model that is valid for all times limits the time wasted to generate network models. Moreover, using a single model that describes the switched RLCM network before and after the switch transition instant might lead to simpler or more generalizable algorithms to finding correct topology after the transition.

6.3 Drawbacks

The main drawback of the approach presented in this chapter is scalability. This approach becomes intractable with large number of switches. This is due to the exponential growth of the number of permutations of switch states with respect to the number of switches.

6.4 Modeling

In chapter 3, the switch characteristic was update to include the derivative of switch variable at the switch transition instant. Using the updated characteristic to construct the complete state space model extended its validity interval to include the switch transition instant.

In this chapter, a similar approach will be used to extend the validity interval even further. A generalization of the Step Independent Source switch model will be introduced. Then, the switch model will be used to construct the promised state space model.

6.4.1 Dependent Source Switch Model

The switch model must preserve the switch characteristic and behave as an ideal switch during any time interval. Such a model can be designed using a dependent voltage or current source with a proper dependence expression. The dependence expression will maintain the ideal behavior of the switch and adapts to represent each topology. In this model, the variability of switch state is represented as dependence expression that is independent to the state space model. This can be compared to the conventional method where the variability is represented by variable switch model, i.e. voltage source for ON switches and current source for OFF switches. Table 6.1 adds the new switch model to Table 2.3.

Table 6.1 Ideal switch model (extended)

Name	ON-State Model	OFF-State Model
SOC	Short circuit	Open Circuit
ZVIS	0 V Voltage source	0 A Current source
SFIS	Step to 0 V voltage source	Step to 0 A current source
Dependent Voltage Source	$v = 0$	$v = expression$ such that $i = 0$
Dependent Current Source	$i = expression$ such that $v = 0$	$i = 0$

6.4.2 Switched Network General State Space Model[8]

The Dependent Source switch model will be used to represent switches in this network model. To construct the network model, we need to choose whether to represent each switch as a dependent voltage source or as a dependent current source. The decision will be made as such to maintain the future work theme of this chapter; therefore, simplicity will be favored.

Switch representation will be chosen as such to allow all capacitor voltages and all inductor currents to be state variables. The choice is not possible for all switched RLCM networks. However, similar to the approach presented in chapter 3, an equivalent network can always be constructed to meet the requirement. Having the requirement met, switch representation can be chosen based on their membership to the tree or co-tree when the edges of the graph are weighted in the following manner:

$$w(VS) < w(C) < w(switch) < w(R) < w(L) < w(CS)$$

Once the switch representation is chosen, the state space model can be constructed using the same approach mentioned in Chapter 2. Table 6.2 summarizes the general state space model, where $1_{condition} = \begin{cases} 0, & condition == false \\ 1, & condition == true \end{cases}$.

---

[8] General State Space Model for short.

Table 6.2 Summary of the Switched Network General State Space Model

	Summary	
Equation form	State equation	$M\,\dot{x} = A\,x + B\,u$
	Output equation	$y = C\,x + D\,u$
Variables	x	Vector of state variables (all capacitor voltages and all inductor currents)
	$\dot{x}$	Vector of first derivatives of state variables
	u	For independent sources, u_i is input
		Dependent voltage source switch model, $u_i = expression * \ 1_{state==OFF}$
		Dependent current source switch model, $u_i = expression * \ 1_{state==ON}$
	y	Vector of outputs variables
Validity interval	$\forall t$ such that $switch\ expressions$ are defined	
Validity requirements	Switch expressions are available Note: if we initialize switch states to be consistent with their model (voltage sources are ON, and current sources are OFF), then any x is a consistent condition	

6.5 Network Analysis

The first step to analyze switched RLCM networks using the general state space model is to construct switch model dependence expressions. To simplify analysis, we let the dependence expressions depend on state variables, input variables (dependent sources), and input derivatives. Because dependence expressions are independent to switch variables, the expressions depend on the topology instead of individual switch states. The advantage of such choice is that the expressions are guaranteed to exist for any topology obeying KVL and KCL. This is because a switch voltage is merely the difference between the voltages at two circuit nodes, and the current is an edge current, both of which can be found by any method of preference (ex: output equations of an RLCM network).

The dependence expressions can be incorporated into the state space model using analysis techniques developed for RLCM networks that include dependent sources. On the other hand, we can utilize the complete state space model to derive equivalent independent sources to replace the dependent sources. Although such derivation can get complicated, the system of equations we obtain can be thought of as an equivalent LTI system. Derivation of equivalent independent sources is possible given the input waveforms, and considering the solution of the complete state space model per topology. This is possible because of our choice of dependence variables.

After reducing the dependencies, discontinuities of state variable are handled automatically in one of two ways as in Table 6.3.

Table 6.3 Discontinuities in Switched Network General State Space Model

Event		How discontinuities are handled
Switch variable becomes zero	**Dependent voltage source turns ON**	Similar to a complete state space model. Implicit re-initialization due to $\frac{d}{dt}var = -var(t_o^-)\,\delta(t-t_o)$
	Dependent current source turns OFF	
Switch variable becomes non-zero	**Dependent voltage source turns OFF**	Impulses responsible for re-initialization appear in the u vector We find the impulse by analyzing the complete state space model corresponding to the topology of interest.
	Dependent current source turns ON	

Steps to determine equivalent independent signals:

- Formulate general state space model. We need to derive i of dependent current source switches and v for dependent voltage source switches.
- Start from initial analysis time (given final conditions) and proceed through the following steps iteratively as you encounter switching events
- At the k^th interval,
    - Given k^th switches states, final conditions form (k-1) interval
        - Update the general state space model to complete state space model then solve it.
    - Given the solution of the complete state space model
        - Resolve dependencies of switch variables of this topology by substituting the solutions, and given inputs.
        - Here you get the value of the switch variables that appear in the state space model during the k^th interval.
    - Given solution of the complete state space model, find final conditions of this interval
        - Here we prepare the final conditions that will be needed in next iteration (k+1)

Example 1 Finding equivalent LTI system

In this example, formulation of an equivalent LTI system will be illustrated. Consider the network presented in Figure 1.1, and consider the following scenario:

- $v_c(0) = v_{co}$
- $S = OFF\ unil\ t_o > 0$
- $S = ON\ from\ t_o\ until\ t_1 > t_o$
- $S = OFF\ starting\ at\ t_1$

To start the analysis, the switch is represented as a Dependent Current Source to obtain the following state equation:

$$\dot{v}_c = -\frac{1}{RC} v_c + \frac{1}{C} i_s$$

Where i_s is constrained as in Table 6.4:

Table 6.4 Switch constraints and dependent source variable

Switch Conduction state	Constraints	i_s
OFF	$i_s = 0$	$i_s = 0$
ON	$v_s = 0$	$i_s = C\,\dot{v}_c + \frac{1}{R}\,v_c$

Given the scenario of interest, the equivalent independent switch model is of the following form:

$$i_s = expression\,(h(t-t_o) - h(t-t_1))$$

The $expression$ can be found by analyzing the complete state space model that corresponds to each configuration the network goes through as in Table 6.5.

Table 6.5 Dependent current source dependence expressions

Configuration	Time interval	i_s
S = OFF	$(\mathbf{0}\ \boldsymbol{t_o})$	$i_s = 0$
S = ON	$[\boldsymbol{t_o}\ \boldsymbol{t_1})$	$i_s = C\,\dot{v}_{vs} + \frac{1}{R}\,v_{vs} + C\,\left(v_{vs}(t_o^-)\ -\ v_{vc}(t_o^-)\right)\delta(t-t_o)$
S = OFF	$[\boldsymbol{t_1}\ \infty)$	$i_s = 0$

Thus we can construct the following expression:

$$i_s(t) = \left\{C\,\dot{v}_{vs} + \frac{1}{R}\,v_{vs} + C\,\left(v_{vs}(t_o^-)\ -\ v_{vc}(t_o^-)\right)\delta(t-t_o)\right\}\ (h(t-t_o) - h(t-t_1))$$

This expression we obtained so far still depends on $v_{vs}(t)$ and $v_c(t_o^-)$.

- $v_{vs}(t)$ is an input, which is known.
- $v_c(t_o^-)$ is a final condition which can be obtained by analyzing the complete state space model of the topology prior to t_o. By doing so, we get the following expression: $v_c(t_o^-) =\ v_{co}\,e^{\frac{-t_o}{RC}}$

Then the Equivalent Independent Source has the following waveform, which depends on known inputs only:

$$i_s(t) = \left\{C\,\dot{v}_{vs} + \frac{1}{R}\,v_{vs} + C\,\left(v_{vs}(t_o^-)\ -\ v_{co}\,e^{\frac{-t_o}{RC}}\right)\,\delta(t - t_o)\right\}\,\left(h(t - t_o) - h(t - t_1)\right)$$

The characteristic of this network in Table 6.6 can be verified with ease.

Table 6.6 Expected characteristic for the network in Figure 1.1 given example scenario

Time	Characteristic
$\boldsymbol{t = t_0}$	$v_c(t_o) = v_{vs}(t_o)$
$[\boldsymbol{t_0}\ \boldsymbol{t_1})$	From state equation: $\dot{v}_c = -\frac{1}{RC}\,v_c + \frac{1}{C}\,i_s = -\frac{1}{RC}\,v_{vs} + \frac{1}{C}\left(C\,\dot{v}_{vs} + \frac{v_{vs}}{R}\right) = \dot{v}_{vs}$ And $v_c(t) = v_{vs}(t)$
$[\boldsymbol{t_1}\ \infty)$	$i_s = 0$ leads to $\dot{v}_c = -\frac{1}{RC}\,v_c$

Although the waveform equation becomes complicated, we can generalize the above derivation for any number of switching events by applying proper time shift techniques and concatenating the result to get the waveform of i_s.

From this example, notice that we are able to use a single LTI model to represent the switched RLCM network. This was possible because the fundamental assumptions used to construct the state space model were never violated. Also notice how impulses caused state discontinuity without requiring us to deal with it in any special way. In some sense, we can consider the impulses in the switch model as the elements that capture the switch non-linearity.

6.6 Network Analysis in the Presence of Internally Controlled Switches and Diodes

We can include internally controlled switches and diodes to the general state space model by including the switch control expression as a set of constraint equations. Then the state space model can be analyzed using constrained model analysis techniques and optimization analysis.

For simple networks, we can find switch transition instants and the correct topology after a switch transition by analyzing the equivalent LTI system model.

Example 2 Including diodes in the general state space model

In this example, we show how the general state space model can be used to construct a set of constrained equations the solution of which is the state of the diode. For this purpose, we replace the switch in Figure 1.1 with a diode. Our aim will be to construct the state space model such that i_s is an input. However, instead of trying to analyze a complete state space model to find an expression for i_s when the switch is ON, we will exploit the fact that $v_s = 0$ when $i_s \neq 0$. Then we attempt to make the equation dependent on switch state in order to use it in finding the consistent switch state

State equation when the diode is represented as a dependent source:

$$\dot{v}_c = \frac{-1}{R\,C}\,v_c + \frac{1}{C}\,i_D$$

Where

$$i_D = \begin{cases} 0 & ,D = OFF \\ some\ expression\ dependent\ on\ v_c(t_o^-) & ,D = ON \end{cases}$$

We need to analyze the complete state space model for the topology prior to the event at t_o in order to find an expression for $v_c(t_o^-)$. Otherwise, we can use the state space model output equation to find an expression for i_D while $v_D = 0$.

$v_D = v_{vs} - v_c = v_{vs} - R\,(i_D - C\,\dot{v}_c)$ and $v_D = 0$

With trivial algebraic manipulation we get:

$$i_D = \frac{1}{R}\,v_{vs} + C\,\dot{v}_c$$

Now we can update our state equation as using i_D.

$$\dot{v}_c = \frac{-1}{R\,C}\,v_c + \frac{1}{C}\,1_{D=ON} * \left(\frac{1}{R}\,v_{vs} + C\,\dot{v}_c\right)$$

Diode control can be expressed as constraint equations:

$$i_s\,v_s = 0$$

$$i_s - v_s \geq 0$$

The system of equations derived in this example, summarized in Table 6.7, models the behavior of the network and maybe optimized to find the consistent diode state without the need to assume continuity of the state variable.

Table 6.7 General state space model as function of switch input

State Equation	$\dot{v}_c = \frac{-1}{R\,C}\,v_c + \frac{1}{C}\,1_{D=ON} * \left(\frac{1}{R}\,v_{vs} + C\,\dot{v}_c\right)$
Constraint Equations	$\left(C\,\dot{x} + \frac{x}{R}\right)\,(u - x) = 0$
	$\left(C\,\dot{x} + \frac{x}{R}\right) - (u - x) \geq 0$
Inputs Vector	$u = [v_{vs}]$
State vector	$x = [v_c]$

6.7 Equivalence between the General and the Complete State Space Models

The complete state space model is a loosely defined general state space model. In the complete state space model, we choose switch representation based on the switch state, and we define the dependence expression only for a small time interval free of switch transitions.

The steps to convert a general state space model to complete state space model are:

- Prepare a general state space model
- For each topology of interest, represent each switch as an open or short circuit enforcing the zero voltage or the zero current for each switch
  - Find the variables of switch terminals that appear in the general state space model (voltages or currents) as function of capacitor voltages, inductor currents, (actual) independent sources, and their derivatives. Note these functions are independent to switch variables.
- Substitute the functions into the general state space model, and reduce it.
  - In certain cases, reduction will eliminate one or more variables from the state vector or the state derivative vector.
  - If a state variable or its derivative is eliminated (but not both), then we must represent the one remaining in the model in terms of the new state variables, all sources (switches modeled as SFIS), and their derivatives.
    - Here, we must keep the derivatives of switch variables.

Example 3 Equivalence between state space models for a buck converter

In this example we show that a general state space model can be reduced to a complete or incomplete state space models by some algebraic manipulations and by using the facts we know about switches ($i = 0$ or $v = 0$). Consider the buck converter in Figure 6.1 where the diode is represented as a switch.

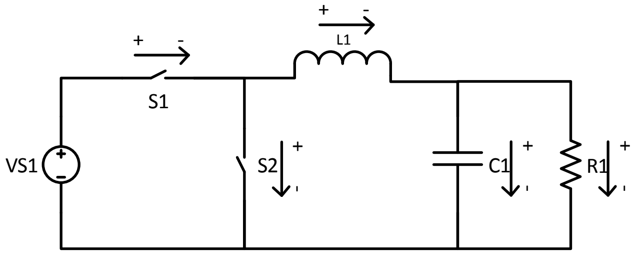


Figure 6.1 Buck converter with diode represented as a switch

The two switches allow the circuit to operate in one of the topologies given in Table 6.8.

Table 6.8 Voltages across and currents through each switch in each of the buck converter topologies

Topology index	Switch	State	voltage	current
1	S1	OFF	$v_{s1} = v_{vs1} - v_{c1} - L_1 \frac{d}{dt} i_{l1}$	$i_{s1} = 0$
	S2	OFF	$v_{s2} = v_{c1} + L_1 \frac{d}{dt} i_{l1}$	$i_{s2} = 0$
2	S1	OFF	$v_{s1} = v_{vs1}$	$i_{s1} = 0$
	S2	ON	$v_{s2} = 0$	$i_{s2} = -i_{l1}$
3	S1	ON	$v_{s1} = 0$	$i_{s1} = i_{l1}$
	S2	OFF	$v_{s2} = v_{vs1}$	$i_{s2} = 0$
4	S1	ON	not a valid circuit[9]	
	S2	ON		

To start the analysis, represent one switch as voltage source and the other as current source so as to make i_l and v_c be states. In this example, S1 is represented as a voltage source arbitrarily as in Figure 6.2. Note that the model is represented as independent sources instead of controlled sources to simplify the figure.

---

[9] When S1 and S2 are both ON, the circuit violates KVL due to a loop containing voltage sources only. It can be assumed that both switches cannot be ON at the same time. This assumption can be removed by introducing a small permanent resister in series with the voltage source.

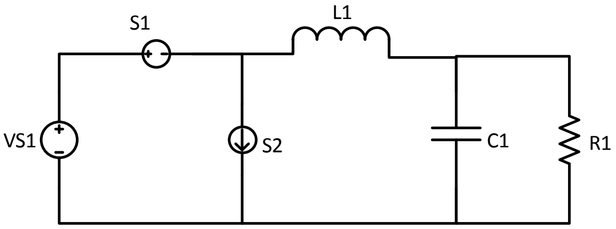


Figure 6.2 Buck converter with the switch and diode represented as sources

The state space model for this network is as follows:

$$\frac{d}{dt}\begin{pmatrix} v_{c1} \\ i_{l1} \end{pmatrix} = \begin{bmatrix} -\frac{1}{R_1C_1} & \frac{1}{C_1} \\ -\frac{1}{L_1} & 0 \end{bmatrix}\begin{pmatrix} v_{c1} \\ i_{l1} \end{pmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ -\frac{1}{L_1} & \frac{1}{L_1} & 0 \end{bmatrix}\begin{pmatrix} v_{s1} \\ v_{vs1} \\ i_{s2} \end{pmatrix}$$

From the equations that have been derived so far, we transfer the general state space model to represent any of the topologies by substituting v_{s1} and i_{s2} according to the desired topology:

- *S1 ON, S2 OFF:*

In this topology, from Table 6.8, $v_{s1} = 0$ and $i_{s2} = 0$ so we get

$$\frac{d}{dt}\begin{pmatrix} v_{c1} \\ i_{l1} \end{pmatrix} = \begin{bmatrix} -\frac{1}{R_1C_1} & \frac{1}{C_1} \\ -\frac{1}{L_1} & 0 \end{bmatrix}\begin{pmatrix} v_{c1} \\ i_{l1} \end{pmatrix} + \begin{bmatrix} 0 \\ \frac{1}{L_1} \end{bmatrix}(v_{vs1})$$

- *S1 OFF, S2 ON:*

In this topology, from Table 6.8, $v_{s1} = v_{vs1}$ and $i_{s2} = -i_{l1}$ so we get

$$\frac{d}{dt}\begin{pmatrix} v_{c1} \\ i_{l1} \end{pmatrix} = \begin{bmatrix} -\frac{1}{R_1C_1} & \frac{1}{C_1} \\ -\frac{1}{L_1} & 0 \end{bmatrix}\begin{pmatrix} v_{c1} \\ i_{l1} \end{pmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ -\frac{1}{L_1} & \frac{1}{L_1} & 0 \end{bmatrix}\begin{pmatrix} v_{vs1} \\ v_{vs1} \\ -i_{l1} \end{pmatrix}$$

$$\frac{d}{dt}\begin{pmatrix} v_{c1} \\ i_{l1} \end{pmatrix} = \begin{bmatrix} -\frac{1}{R_1C_1} & \frac{1}{C_1} \\ -\frac{1}{L_1} & 0 \end{bmatrix}\begin{pmatrix} v_{c1} \\ i_{l1} \end{pmatrix} + \begin{pmatrix} 0 \\ -\frac{v_{vs1}}{L_1} + \frac{v_{vs1}}{L_1} \end{pmatrix}$$

$$\frac{d}{dt}\begin{pmatrix} v_{c1} \\ i_{l1} \end{pmatrix} = \begin{bmatrix} -\frac{1}{R_1C_1} & \frac{1}{C_1} \\ -\frac{1}{L_1} & 0 \end{bmatrix}\begin{pmatrix} v_{c1} \\ i_{l1} \end{pmatrix}$$

- *S1 OFF, S2 OFF:*

In this topology, from Table 6.8, $v_{s1} = v_{vs1} - v_{c1} - L_1\frac{d}{dt}\, i_{l1}$ and $i_{s2} = 0$ so we get

$$\frac{d}{dt}\begin{pmatrix} v_{c1} \\ i_{l1} \end{pmatrix} = \begin{bmatrix} -\frac{1}{R_1C_1} & \frac{1}{C_1} \\ -\frac{1}{L_1} & 0 \end{bmatrix}\begin{pmatrix} v_{c1} \\ i_{l1} \end{pmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ -\frac{1}{L_1} & \frac{1}{L_1} & 0 \end{bmatrix}\begin{pmatrix} v_{vs1} - v_{c1} - L_1\frac{d}{dt}\, i_{l1} \\ v_{vs1} \\ 0 \end{pmatrix}$$

$$\frac{d}{dt}\begin{pmatrix} v_{c1} \\ i_{l1} \end{pmatrix} = \begin{bmatrix} -\frac{1}{R_1C_1} & \frac{1}{C_1} \\ -\frac{1}{L_1} & 0 \end{bmatrix}\begin{pmatrix} v_{c1} \\ i_{l1} \end{pmatrix} + \begin{bmatrix} 0 \\ -\frac{v_{vs1} - v_{c1} - L_1\frac{d}{dt}\, i_{l1}}{L_1} + \frac{v_{vs1}}{L_1} \end{bmatrix}$$

$$\frac{d}{dt}\begin{pmatrix} v_{c1} \\ i_{l1} \end{pmatrix} = \begin{bmatrix} -\frac{1}{R_1C_1} & \frac{1}{C_1} \\ -\frac{1}{L_1} & 0 \end{bmatrix}\begin{pmatrix} v_{c1} \\ i_{l1} \end{pmatrix} + \begin{bmatrix} 0 \\ \frac{v_{c1} + L_1\frac{d}{dt}\, i_{l1}}{L_1} \end{bmatrix}$$

$$\frac{d}{dt}\binom{v_{c1}}{i_{l1}} = \begin{bmatrix} -\frac{1}{R_1C_1} & \frac{1}{C_1} \\ -\frac{1}{L_1} & 0 \end{bmatrix}\binom{v_{c1}}{i_{l1}} + \begin{pmatrix} 0 \\ \frac{1}{L_1} \end{pmatrix}(v_{c1}) + \binom{0}{1}\frac{d}{dt}(i_{l1})$$

Note at this point that $\frac{d}{dt}i_{l1}$ cancels from the sides of the equation implying that this variable is no longer a state and thus the state space dimensionality becomes one. Therefore, we have to update the state vector and remove i_{l1} from it. To do so, we need to find the equation of i_{l1} with respect to all input variables (including switch variables) and the new state variables:

$$i_{l1} = i_{s1} - i_{s2}$$

Thus we get

$$\frac{d}{dt}\binom{v_{c1}}{0} = \begin{bmatrix} -\frac{1}{R_1C_1} & \frac{1}{C_1} \\ -\frac{1}{L_1} & 0 \end{bmatrix}\binom{v_{c1}}{i_{s1} - i_{s2}} + \begin{pmatrix} 0 \\ \frac{1}{L_1} \end{pmatrix}(v_{c1})$$

$$\frac{d}{dt}\binom{v_{c1}}{0} = \begin{bmatrix} -\frac{1}{R_1C_1} & \frac{1}{C_1} \\ 0 & 0 \end{bmatrix}\binom{v_{c1}}{i_{s1} - i_{s2}}$$

$$\frac{d}{dt}\binom{v_{c1}}{0} = \begin{bmatrix} -\frac{1}{R_1C_1} & \frac{1}{C_1} \\ 0 & 0 \end{bmatrix}\binom{v_{c1}}{i_{s1} - i_{s2}}$$

But, from Table 6.8, $i_{s1} = 0$ and $i_{s2} = 0$, so we finally get

$$\frac{d}{dt}(v_{c1}) = -\frac{v_{c1}}{R_1C_1}$$

Chapter 7 Conclusion

This thesis discusses an ideal switched RLCM network simulation application from a mathematical theory and from a software design perspectives. The project was based on state space modeling and analysis, which is appropriate for understanding system level behavior of power converters. The software complexity necessary to develop the simulator was addressed by employing object oriented programming tools; namely C++ programming language and the unified modeling language (UML).

Ideal switched RLCM networks is analyzed using piecewise linear analysis techniques. In each piecewise linear time interval, the network is represented as a linear RLCM network that can be modeled and analyzed using state space methods. In the analysis of power converters, this approach is more appropriate than companion model and modified nodal analysis. Using this approach, the software can recognize changes in network topology, which is a fundamental trait of ideal switched RLCM networks. Moreover, tools developed for this software is useful beyond numerical simulation.

The effects of ideal switch extreme non-linearity must be analyzed careful to simulate switched RLCM networks correctly. This thesis approaches the problem differently from literature. Unlike literature, state re-initialization, impulse presence, and impulse propagation are addressed as modeling problems instead of analysis problems. This improves the computational efficiency and simplifies searching for correct topologies.

References

[1] F. C. Y. Lee and S. Kelkar, "A fast time domain digital simulation technique for power converters: Application to a buck converter with feedforward compensation," *Power Electronics, IEEE Transactions on,* Vols. PE-1, no. 1, pp. 21 - 31, Jan 1986.

[2] F. Lee and Y. Yu, "Computer-Aided Analysis and Simulation of Switched DC-DC Converters," *Industry Applications, IEEE Transactions on,* Vols. IA-15, no. 5, pp. 511 - 520, Sept 1979.

[3] A. Luciano and A. Strollo, "Electronics, IEEE Trans ...> Volume:5 Issue:3 Help Working with Abstracts," *Power Electronics, IEEE Transactions on,* vol. 5, no. 3, pp. 363 - 370, Jul 1990.

[4] D. Bedrosian and J. Vlach, "Time-domain analysis of networks with internally controlled switches," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on,* vol. 39, no. 3, pp. 199-212, Mar 1992.

[5] R. Frasca, M. Camlibel, I. Goknar, L. Iannelli and F. Vasca, "Linear Passive Networks With Ideal Switches: Consistent Initial Conditions and State Discontinuities," *Circuits and Systems I: Regular Papers, IEEE Transactions on,* vol. 57, no. 12, pp. 3138-3151, Dec 2010.

[6] N. Femia, G. Spagnuolo and M. Vitelli, "Unified analysis of synchronous commutations in switching converters," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on,* vol. 49, no. 8, pp. 1150-1166, Aug 2002.

[7] L. O. Chua and P.-M. Lin, Computer-aided analysis of electronic circuits: algorithms and computational techniques, Prentice-Hall, 1975.

Appendix A Parameters for Test Cases

Test Case 1 Switched Active RC Network

Table A.1 Test Case 1 network parameters

Device	Parameter	Value	unit
R1	R	200	Ω
C1	C	0.001	F
C1	V_{co}	0	V
V1	V	5	V DC

Test Case 2 Switched Capacitor Loop

Table A.2 Test Case 2 network parameters

Device	Parameter	Value	unit
C1	C	0.001	F
C1	V_{co}	20	V
C2	C	0.002	F
C2	V_{co}	5	V

Test Case 3 Bridge Rectifier Network

Table A.3 Test Case 3 network parameters

Device	Parameter	Value	unit
C1	C	4E-3	F
C1	V_{co}	157.5050	V
L1	L	1.1E-3	H
L1	I_{Lo}	0	A
V1	V	120	V RMS
V1	Frequency	60	Hz
I1	I	5	A DC

Test Case 4 Buck Converter with constant switching frequency

Table A.4 Test Case 4 network parameters

Device	Parameter	Value	unit
R1	R	10	Ω
C1	C	1E-3	F
C1	V_{co}	0	V
L1	L	2E-3	H
L1	I_{Lo}	0	A
V1	V	10	V DC
-	D	0.8	Percent ON

Test Case 5 Buck Converter with feedback control system

Table A.5 Test Case 5 network parameters

Device	Parameter	Value	unit
R1	R	18	Ω
C1	C	100E-6	F
C1	V_{co}	148	V
L1	L	100E-6	H
L1	I_{Lo}	7	A
V1	V	170	V DC
-	vRef	150	V DC
-	α	2.35294118	-
-	β	0.1	-
Integrator	Initial value	0.8	-
Saw-tooth Generator	Frequency	50E3	Hz

Appendix B ISM and SCN Files for Test Cases

Test Case 1 Switched Active RC Network

File 1 test_01_simple_case.ism

```
//: test_01_simple_case.sim

/*****************************************************************************
Test Case 1: Switched Active RC Network
*****************************************************************************/

/*****************************************************************************
2014-01-22 Saleh Albeaik Created
*****************************************************************************/

// Define nodes.
Node GND;
Node N1;
Node N2;
Node N3;

// Define resistors.
Resistor R1 from(N3) to(GND) R(200);

// Define capacitors.
Capacitor C1 from(N2) to(GND) C(.001);

// Define voltage sources.
VoltageSource V1 from(N1) to(GND) SignalGenerator.Function(const) SignalGenerator.DC(5);

// Define Short Circuits
VoltageSource V2 from(N2) to(N3) SignalGenerator.Function(const) SignalGenerator.DC(0);

// Define Switches
Switch S1 from(N1) to(N2) ControlMethod.State(0);
```

File 2 test_01_simple_case.scn

```
//: test_01_simple_case.scn

init;

transition mode(dynamic);

modify at(.1) S1.ControlMethod.State(1);

modify at(.2) S1.ControlMethod.State(0);

record at(0.001) every(0.001) t
               V1.v
               C1.v
        ;


advance by(1.5);

exit;
```

Test Case 2 Switched Capacitor Loop

File 3 test_02_simple_case.ism

```
//: test_02_simple_case.sim

/******************************************************************************
Test Case 2: Switched Capacitor Loop
******************************************************************************/

/******************************************************************************
2014-01-22 Saleh Albeaik Created
******************************************************************************/

// Define nodes.
Node GND;
Node N1;
Node N2;

// Define capacitors.
Capacitor C1 from(N1) to(GND) C(.001) V0(20);
Capacitor C2 from(N2) to(GND) C(.002) V0(5);

// Define Switches
Switch S1 from(N1) to(N2) ControlMethod.State(0);
```

File 4 test_02_simple_case.scn

```
//: test_02_simple_case.scn

init;

transition mode(dynamic);

modify at(.5) S1.ControlMethod.State(1);

modify at(1) S1.ControlMethod.State(0);

record at(0.001) every(0.001) t
               C1.v
               C2.v
               S1.v
        ;


advance by(1.5);

exit;
```

Test Case 3 Bridge Rectifier Network

File 5 test_03_rectifier.ism

```
//: test_03_rectifier.sim

/*******************************************************************************
Test Case 3: Bridge Rectifier Network
Test of a  rectifier with a smoothing filter
*******************************************************************************/

/*******************************************************************************
2014-01-27 Saleh Albeaik Created
*******************************************************************************/

// Define nodes.
Node GND;
Node N1;
Node N2;
Node N3;
Node N4;
Node N5;

// Define Load.
CurrentSource I1 from(N5) to(N3) SignalGenerator.Function(const) SignalGenerator.DC(5);

// Short Circuit
VoltageSource V2 from(N4) to(N5) SignalGenerator.Function(const) SignalGenerator.DC(0);

// Define capacitors.
Capacitor C1 from(N4) to(N3) C(4e-3) V0(157.5050);

// Define inductors.
Inductor L1 from(N1) to(N2) L(1.1e-3) I0(0);

// Define voltage sources.
VoltageSource V1 from(N1) to(GND) SignalGenerator.Function(sin) SignalGenerator.Amplitude(340)
SignalGenerator.Frequency(60) SignalGenerator.Phase(0) SignalGenerator.DC(0);

// Define diodes
Diode D4 from(N3) to(N2);
Diode D2 from(N3) to(GND);
Diode D1 from(N2) to(N4);
Diode D3 from(GND) to(N4);
```

File 6 test_03_rectifier.scn

```
//: test_03_rectifier.scn

init;

transition mode(dynamic);

record every(0.00002) t
               C1.v
               C1.i
               I1.i
               L1.v
               V1.v
               V1.i
               C1.i
               D1.v
               D3.v
        ;


advance by(0.1);

exit;
```

Test Case 4 Buck Converter with constant switching frequency

File 7 test_04_buck_constant_switching.ism

```
//: test_04_buck_constant_switching.sim

/*******************************************************************************
Test Case 4: Buck Converter with constant switching frequency
*******************************************************************************/

/*******************************************************************************
2014-01-22 Saleh Albeaik Created
*******************************************************************************/

// Define nodes.
Node GND;
Node N1;
Node N2;
Node N3;
Node N4;
Node N5;

// Define resistors.
Resistor R1 from(N5) to(GND) R(10);

// Define capacitors.
Capacitor C1 from(N4) to(GND) C(1e-3);

// Define inductors.
Inductor L1 from(N3) to(N4) L(2e-3);


// Define voltage sources.
VoltageSource V1 from(N1) to(GND) SignalGenerator.Function(const) SignalGenerator.DC(10);

// Short Circuit
VoltageSource V2 from(N4) to(N5) SignalGenerator.Function(const) SignalGenerator.DC(0);

// Define Switches and diodes
ControlledSwitch S1 from(N2) to(N3) ControlMethod.ControlTerminal(dsink1);
Diode D1 from(N1) to(N2);
Diode D2 from(GND) to(N3);

//------------------------------------------------------------------
//--| Define Control System |--------------------------------------

SignalGenerator sgn1 SignalGenerator.Function(sawtooth) SignalGenerator.Frequency(1e3)
SignalGenerator.T0(0) SignalGenerator.LowValue(0) SignalGenerator.HighValue(1);

DigitalSink dsink1 Input(sgn1) Threshold(.2);
```

File 8 test_04_buck_constant_switching.scn

```
//: test_04_buck_constant_switching.scn

init;

transition mode(dynamic);

record at(0) every(0.00005) t
               C1.v
               L1.i
               V1.v
        ;


advance by(0.4);

exit;
```

Test Case 5 Buck Converter with feedback control system

File 9 test_06_buck_closed_loop.ism

```
//: test_06_buck_closed_loop.sim

/******************************************************************************
Test Case 5: Buck Converter with feedback control system
- single pole control system
******************************************************************************/

/******************************************************************************
2014-01-22 Saleh Albeaik Created
******************************************************************************/

// Define nodes.
Node GND;
Node N1;
Node N2;
Node N3;
Node N4;

// Define resistors.
Resistor R1 from(N4) to(GND) R(18);

// Define capacitors.
Capacitor C1 from(N3) to(GND) C(100e-6) V0(148);

// Define inductors.
Inductor L1 from(N2) to(N3) L(100e-6) I0(7);


// Define voltage sources.
VoltageSource V1 from(N1) to(GND) SignalGenerator.Function(const) SignalGenerator.DC(170);

// Short Circuit
VoltageSource V2 from(N3) to(N4) SignalGenerator.Function(const) SignalGenerator.DC(0);

// Define Switches and diodes
ControlledSwitch S1 from(N1) to(N2) ControlMethod.ControlTerminal(dsink1);
Diode D1 from(GND) to(N2);

//-----------------------------------------------------------------
//--| Define Control System |--------------------------------------

// Define sensors
Sensor sensor1 Input(R1) Type(voltage);

// Define inverters using gain blocks
GainBlock invSensor1 Input(sensor1) Gain(-1);
GainBlock invGb2 Input(gb2) Gain(-1);

// Define adders
Adder add1 Input_1(vRef) Input_2(invSensor1);
Adder add2 Input_1(gb1) Input_2(invGb2);

// Define gain blocks
GainBlock gb1 Input(add1) Gain(2.35294118);  //k over tou
GainBlock gb2 Input(intg1) Gain(0.1); //1 over tou

// Define integrators
Integrator intg1 Input(add2) X0(0.8);

// Define signal generators
SignalGenerator vRef SignalGenerator.Function(const) SignalGenerator.DC(150);
SignalGenerator sgn1 SignalGenerator.Function(sawtooth) SignalGenerator.Frequency(50e3)
SignalGenerator.T0(0) SignalGenerator.LowValue(0) SignalGenerator.HighValue(1);
```

```
// Define comparators
Comparator cmp1 Input_2(sgn1) Input_1(intg1) GreaterThanOutput(1) LessThanOutput(-1);

// Define digital sinks
DigitalSink dsink1 Input(cmp1) Threshold(0);
```

File 10 test_06_buck_closed_loop.scn

```
//: test_06_buck_closed_loop.scn

init;

transition mode(dynamic);

record at(0) every(0.000001) t
               C1.v
               L1.i
               V1.v
               vRef.result
        ;


advance by(0.005);

exit;
```